Université Catholique de Louvain
Faculté des Sciences Appliquées
Laboratoire de Microélectronique

Modelling and Security Analysis of Authenticated Group Key Agreement Protocols

Olivier Pereira

Thèse soutenue en vue de l'obtention du grade de Docteur en Sciences Appliquées

Composition du jury:

Pr. J.-J. Quisquater (UCL - DICE) – Promoteur
Dr. J. D. Guttman (The MITRE Corporation – USA)
Pr. A. van Lamsweerde (UCL - INFO)
Pr. O. Markowitch (Université Libre de Bruxelles)
Pr. S. A. Schneider (Royal Holloway - University of London – UK)

Pr. J.-D. Legat (UCL - DICE) – Président

Louvain-la-Neuve, Belgique
2003

Abstract

Authenticated Group Key Agreement Protocols are protocols allowing a group of principals to contributively generate a key by the exchange of messages on a network possibly controlled by an attacker. Furthermore, their execution also guarantees all group members that the key they obtained can only be known by the other intended protocol participants. These protocols can be exploited in many applications such as audio or videoconferencing, replicated servers (such as database, web, time servers), chat or network games for instance.

AGKAP's present several particularities that make them interesting case studies for research in the theory of security. At first, the consideration of the number of protocol participants as a parameter raises several complexity problems that are not present in the classical two or three-party frameworks. Furthermore, up to now, the security properties of group protocols have roughly been considered as direct extensions of two-party properties, what does not capture several plausible attack scenarios. A second interesting aspect of the analysis of AGKAP's is the consideration of Diffie-Hellman-type primitives, that present properties out of the scope of most classical models.

We started our study with the construction of a simple model for the analysis of a classical family of protocols: the Cliques AGKAP's. This allowed us to discover several attacks and define different flavors of group security properties. We then tried to fix these protocols, what led us to extend our model in order to prove that it is in fact impossible to build a secure AGKAP based on the same design assumptions as the Cliques protocols. Finally, we designed a new AGKAP based on different cryptographic primitives (signature and hash functions) for which we proved authentication, freshness and secrecy properties. A comparison with a similar AGKAP developed in parallel to ours is also proposed.

Acknowledgements

First of all, I would like to thank Jean-Jacques Quisquater for having accepted to be my advisor for this thesis. His suggestions, encouragements were always a precious support, as well as the freedom of action he allowed me.

I also would like to thank Joshua Guttman, Axel van Lamsweerde, Olivier Markowitch and Steve Schneider and for having accepted to be members of my jury: their remarks were an invaluable help. I am also grateful to Jean-Didier Legat for having agreed to preside this jury.

Working with the UCL Crypto Group members was of great help and pleasure. I would like to thank more particularly Damien Giry for his friendship and support, Francesco Sica for his precious indications, Gaël Hachez who patiently answered my "questions du jour", Olivier Chevassut who challenged me with the Cliques protocols and, last but not least, Sylvie Baudine for her availability, efficiency, and patient proofing of this text.

This research also greatly benefitted from the excellent working conditions offered by the Belgian "Fonds National de la Recherche Scientifique" (FNRS). I would like to thank them for their financial support.

As always, my family and friends offered me a continuous support, each of them in their own way. More specifically, Bénédicte, François and Frédéric allowed me to improve different parts of this text. I am also grateful to Jacques and Nicolas who welcomed me during an important part of this thesis writing-up phase.

Several anonymous referees provided many useful suggestions about selected parts of this thesis material. I would like to thank them for doing this important work so conscientiously and thoughtfully.

Contents

Abstract iii

Acknowledgements v

Notations ix

Introduction 1
1. Modelling and Analysis of Security Protocols 1
2. Authenticated Group Key Agreement Protocols 4
3. Organization of the Work 5

Chapter 1. Analysis of Cliques Protocols 7
1. Introduction 7
2. The A-GDH.2 Protocol 7
3. Intended Security Properties 11
3.1. Implicit Key Authentication 11
3.2. Perfect Forward Secrecy 11
3.3. Resistance to Known-Key Attacks 12
4. A Model for the Analysis of the Cliques Protocols 13
4.1. Introduction and Related Works 13
4.2. Messages and Intruder's Knowledge 15
4.3. Intruder Capabilities 17
4.4. Proving Security Properties 18
5. Analysis of the A-GDH.2 Protocol 25
5.1. Implicit Key Authentication 25
5.2. Perfect Forward Secrecy 29
5.3. Resistance to Known-Key Attacks 31
6. Considering the A-GDH.2-MA Protocol 34
6.1. Definition of the A-GDH.2-MA Protocol 34
6.2. Analysis of the A-GDH.2-MA Protocol 35
7. Analysis of the SA-GDH.2 Protocol 37
8. Concluding Remarks 40

Chapter 2. A Fix for the A-GDH Protocols? 41
1. Introduction 41
2. A Family of Protocols 42
3. Properties of GDH-Protocols 50

4. Expression of $R_i \cdot K_i$ as a Product of Contributions 53
5. Building Attacks Against GDH-Protocols 58
5.1. Building Simple Pairs of Elements of G 59
5.2. Combining Pairs of Elements of G 64
5.3. Obtaining a Secret Pair 69
6. Concluding Remarks 78
6.1. Summary 78
6.2. Discussion of the Results 80
6.3. Conclusion 83

Chapter 3. Design and Analysis of an AGKAP 85
1. Introduction 85
2. Basic Scheme and Security Requirements 85
3. Theoretical Background 86
3.1. Strand Spaces and Bundles 86
3.2. Authentication Tests 88
3.3. Recency 89
4. Construction of a New Protocol 90
4.1. Introduction 90
4.2. Modelling of the GDH.2 Protocol 90
4.3. Authentication Services 91
4.4. Secrecy 93
4.5. Recency 94
5. Correctness of the AT-GDH Protocol 95
5.1. The AT-GDH Protocol 95
5.2. Achieving Implicit Key Authentication 97
5.3. Achieving Resistance to Known Session-Secret Attacks 99
5.4. Achieving Individual Forward Secrecy 101
6. Comparison with the AKE1 Protocol 101
6.1. The AKE1 Protocol 101
6.2. Inclusion of the Full Group Constitution 102
6.3. Recency Properties 103
6.4. Definition of the Group Key 105
6.5. Computational Considerations 106
6.6. Efficiency Considerations 107
7. Concluding Remarks 107

Conclusion 109

Bibliography 113

Appendix A. Publications list 119

Appendix B. Building an Attack from a Linear System Solution 121

Appendix C. Illustration of Chapter 2's Attack Process 125

Notations

Operators and Relations

$\sqsubset$	Subterm relation
$\prec$	Precedence relation
$g[i]$	Sequence g of i elements
$g(i)$ or $(g)_i$	i-th element of the sequence g
$\langle s, j\rangle$	j-th node of the strand s
$\langle \alpha_i, j\rangle$	$uns\_term(n_j)(i_j)$ where (n_j, i_j) is the j-th element of the history α_i
$[\mathsf{S}\backslash M_I : p]$	Value that p would have if it was computed in a session where all principals included in S are replaced by M_I
$\{\!\lvert m \rvert\!\}_{S_i}$	Signature of the message m through M_i's long-term signing key
$\mathcal{H}(m)$	Hash of the message m
p_{R}	Random contribution part of $p \in \mathsf{P}$
p_{K}	Key part of $p \in \mathsf{P}$ ($p = p_{\mathsf{R}} \cdot p_{\mathsf{K}}$)

Variables

A	Algebra of messages
$\alpha \in \mathcal{G}$	Public generator of $\mathcal{G}$
$\alpha \in \mathsf{G}$	Unit element of G
α_i	History of the element that M_i uses to compute his view of the group key
E	$\mathsf{R} \cup \mathsf{K}$
E_I	Set of elements of E known by M_I
$\mathcal{G}$	Multiplicative cyclic group
G	Multiplicative group used to model $\mathcal{G}$
K	Set of long-term symmetric keys
K_i	Set of long-term symmetric keys known by M_i
K_I	Set of long-term symmetric keys known by M_I
K_i	Key part of the value that M_i uses to compute his view of the group key
K_{ij}	Long-term symmetric key shared by M_i and M_j

M	Group of (honest) principals
$M_1, \ldots, M_n$	Members of M
M_I	Intruder (also referred to as "Attacker", "Adversary" or "Penetrator")
n	Cardinality of M
P	Commutative group freely generated from E
P_I	Set of elements of P known by M_I
P_S	Set of elements of P representing the ratio of the powers of α in secret pairs of elements of G
R	Set of random contributions to the group key
R	$\prod_{M_i \in \mathsf{M}} r_i$
r_i	Random contribution generated by M_i
R_i	Random part of the value M_i uses to compute his view of the group key
S	Set of available services
S_i	Long-term signing key of M_i

Introduction

The title of this thesis is made up of two parts: "modelling and security analysis" and "authenticated group key agreement protocols".

A first aspect of this thesis will therefore be the development of a method for modelling and analyzing a set of security protocols. The second aspect will be the application of this method to the analysis of a family of authenticated group key agreement protocols. We successively consider these two facets in the following sections.

1. Modelling and Analysis of Security Protocols

One proposed definition of "security protocol" (also known as "cryptographic protocols" in the literature) is "a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective" ([53], p. 33). These actions are typically the sending and reception of messages that are built by exploiting some cryptographic primitives (such as hashing functions, encryption or signature schemes). Some internal operations are also usually specified, in order to compute a session key from the messages previously received by another entity for instance. The goals of these protocols are manifold: establishing sessions keys between entities, ensuring freshness, secrecy, non-repudiation, authenticating one entity to an other, ...

The design and analysis of these protocols have been recognized as much more difficult than initially thought: in [26], Donovan & al. are showing that more than half of the fifty one protocols inventoried in a classical library [22] are flawed. Several reasons explain this fact.

First, protocols are designed to be used in specific environments (internet, local networks, dial-up connections, ...), the properties of which are usually complex to define precisely. These environments directly influence the ways the intruder (also referred to as "attacker", "adversary" or "penetrator") will interact with the honest parties in order to undermine the different security properties. Furthermore, depending on the cryptographic primitives exploited, a number of assumptions have to be stated upon the computational capabilities of the intruder: they are sometimes considered as infinite but, in most cases, cryptographers are

assuming the hardness of some arithmetic problems. All these assumptions and environmental conditions will lead the designer of security protocols to develop models, with different degrees of realism according to the context. Since these models are always approximations, they will typically be enhanced as the research progresses and new kinds of attacks are discovered.

After having defined a model, the security analyst still has to prove that the protocol allows reaching the required security goals, at least in the context defined in the model. However, the definition of these security properties has also appeared to be a much more complicated task than initially supposed: properties a priori as simple as authentication or secrecy turned out to be expressible in many different flavors (inside some model and between different models) and it is often difficult to relate the specifications in one model into those in another [2, 11, 30, 46]. Furthermore, proof techniques widely differ from one model to the other, each one presenting benefits and disadvantages.

Finally, the use of security protocols becoming widespread, more and more complex protocols (multicast, e-commerce, ...) were designed. This naturally came with security properties more difficult to specify and imposed the use of more expressive models, what typically also resulted in much more complex proofs.

During the last twenty years, two families of models for the analysis of security protocols have been developed in two little related communities.

In one of these families of models, to which we refer as *computational models*, the analysis of security protocols is approached as a branch of complexity theory. The intruder is considered as a probabilistic polynomial-time (PPT) algorithm that exploits oracles representing the different participants to the analyzed protocol. The different cryptographic primitives exploited (e.g. hashing functions, encryption or signature schemes) are considered as algorithms mapping bit-strings into bit-strings and the proven security properties typically assert that bad things have (very) low probability to happen. This is usually established by proving that a PPT algorithm undermining a security property with a non-negligible probability can also be used to gain a non-negligible advantage in solving some reputed hard problem (integer factorization, discrete logarithm extraction, ...).

The use of this kind of model has advantages and drawbacks. On the one hand, the low level of abstraction adopted makes the security proofs obtained in these models very convincing. Nevertheless, we still have to keep in mind that a number of idealizations are still stated: these models consider that the only ways the intruder has to interact with the participants are those described in the definition of the requests that

can be performed to the oracles, what does not take into account the large family of "side-channel" attacks (i.e. timing attack [43], DPA [44], ...) for instance. Furthermore, several assumptions are typically stated about the quality of the implementation of cryptographic primitives. As a consequence, attacks against elements such as random or prime number generators are almost never taken into account. Nevertheless, from a general point of view, the computational adversary remains much stronger than the one considered in the other family of approaches. This advantage has a cost: the security proofs obtained are very complex and their adaptation to similar protocols is difficult: they roughly remain only practicable by specialists.

The other family of models, to which we refer as *logical models*, considers the protocols from a much higher abstraction level, and finds its roots in positions adopted by Needham and Schroeder in [59] and Dolev and Yao in [25].

In these models, messages are symbolically considered and are elements of well defined algebras: the atomic elements are typically principal identifiers, keys and nonces (i.e. hard to guess random values generated during the execution of a protocol). These elements are usually combined by means of concatenation and encryption, typically assuming that the message algebra is free. The (honest) participants will be able to take part a certain number of sessions of the protocol in parallel, the intruder being able to take part in any of them. All messages exchanged during protocol executions are routed through a network that is assumed to be completely under the control of the intruder, who can record, delete, replay, reroute and reorder messages. This intruder is also able to create new messages, as long as he does not use keys he does not know. So, he will behave as a non-deterministic process that can send any message he can generate to any participant at any time and intercept any message sent by the participants.

The security properties are typically expressed as logical propositions on sessions and elements of the message algebra (for instance: "If Alice completes a session of the protocol, apparently with Bob, session during which a key K_{AB} has been exchanged, then Bob has completed a session of the protocol, apparently executed with Alice, and has obtained the same key").

The conception of the intruder in this kind of approach is much weaker than the one of computational approaches: the message algebra is assumed to be free, which is obviously not the case in computational models where a message, be it a cleartext or a ciphertext, is expressed as a string of bits. Furthermore, logical approaches typically do not take into account the properties of the cryptographic primitives

exploited, such as the chaining mode in block-ciphers [1] or the multiplicative property of RSA [66] (exceptions may however be found in [62, 75] for instance). Such idealizations however confer a number of advantages to these methods: in many cases, the security proofs can be automated or transposed to similar protocols [12, 47, 48, 70], and are therefore much easier to manage by non-specialists [26]. Furthermore, this simplicity allowed the analysis of complex protocols [8, 51, 57, 68], the study of which would have been excessively difficult in a computational framework.

A number of works aiming at relating these two kinds of approach have been presented during the last three years [4, 34, 37, 54, 56, 65]. The main goal of these works is the enjoyment of the best aspects of the two worlds: the low abstraction level of computational methods, and the simplicity of logical ones. Typically, these works establish that the existence of a proof of some security properties in a logical framework (variants of the spi-calculus [3], strand spaces [78], ...) implies the existence of the proof of a similar property in a computational framework. This comes down to prove that a computational intruder cannot behave differently than a logical intruder with a non-negligible probability.

2. Authenticated Group Key Agreement Protocols

In the previous section, we mentioned that the more and more widespread use of security protocols is accompanied by the emergence of the need of diverse new protocol types. Authenticated group key agreement protocols (AGKAP for short) may be placed in this category.

Basically, an AGKAP is a protocol enabling a group of users M of moderate size (typically less than one hundred members) to contributively generate a group key that can only be known by the members of the considered group. These protocols are useful for audio- or video-conferencing, for replicated servers (such as database, web, time servers), chat or network games for example.

More precisely, AGKAP are key agreement protocols, rather than key distribution protocols. This means that the group members are generating a group key in a contributive way, so that none of them is able to predetermine its value. Typically, each group member will provide a random value, and the group key will be a function of all these contributions. Key agreement protocols are usually more expensive in resources (bandwidth, computational requirements, ...) than key distribution protocols. However, they are usually preferable from the reliability and security point of view: they do not require the presence of a trusted server, which avoids to provide a designated target to the intruder, target whose compromise would typically result in a security failure for all groups or in a denial of service. Besides, the constitution of

the groups should be dynamically modifiable without a compulsory replay of a whole session of the key agreement protocol. This implies that, in addition to the group key agreement protocol itself, sub-protocols allowing member addition or deletion will be defined in order to manage these changes.

A number of group key agreement protocols have been published during the last years [7, 13, 14, 15, 19, 39, 40, 41, 42, 64, 71, 73, 74, 79]. Several authors do not consider authentication as an objective for their protocol and assume the communication channels to be authenticated. Hence, when a proof of their security is given, it is against a passive adversary that is only able to eavesdrop the messages exchanged on the network.

In most of the cases, the security of AGKAP's is only justified via informal arguments: some computational models have only recently (since 2001) been proposed for the analysis of AGKAP [17, 72], in parallel with the logical model [63] we are presenting in this thesis.

As described in [50, 51, 67], the development of a logical model allowing the analysis of AGKAP presents several particularities. First, the protocols considered in the classical logical models usually involve a very limited and well-defined number of participants (2 or 3). Group protocols, which involve an a priori unknown number of participants, therefore raises new problems. Furthermore, AGKAP being contributive protocols, they usually involve different extensions of the classical two-parties Diffie-Hellman protocol [23]. Their analysis will therefore require (for some of them at least) the modelling of arithmetic properties (commutativity, associativity, ...) at a lower level of abstraction than the one usually considered in this kind of approach.

3. Organization of the Work

The next parts of this work are organized as follows. In Chapter 1, we consider a family of AGKAP proposed in the context of the Cliques project [6, 7]. In that chapter, we propose a model of rather simple exploitation allowing the analysis of these protocols, what allowed us to pinpoint several attacks against each of the Cliques AGKAP, and against each of the claimed security properties (or against a slight and plausible variant for one of them).

In Chapter 2, we examine how these protocols should be fixed, i.e. how we could modify them in such a way that they would present the claimed security properties and preserve the fundamental principles of their structure (otherwise it could not really be called a fix). This question will lead us to refine the model presented in Chapter 1, what renders

its use less intuitive, but allows us to precisely define the class of protocols we consider as well as the properties they must present due to their structure. We then prove that it is impossible to write a protocol of this class that would present the (implicit) key authentication property. This proof is established by showing a systematic way to build an attack against any protocol of this family.

Given that result, in Chapter 3, we turn to the design of a variant of the GDH.2 protocols presented in the Cliques project, variant exploiting a hashing function and a signature scheme. The use of the design strategy suggested in [33] will lead us to build a protocol very similar in many aspects to those proposed by Bresson & al. in [13, 14] and prove its security in the strand space paradigm. We finally examine the differences between our protocol and those presented in [13, 14], what will allow us to illustrate the different benefits and disadvantages of logical and computational approaches.

We do not provide a detailed overview of the area of logical analysis of security protocol: we suggest the reader interested in such readings to refer to the book of P. Ryan and S. Schneider [67], to the tutorial lectures given at FOSAD in 2000 [31] or to the recent state of the art of C. Meadows [52].

CHAPTER 1

Analysis of Cliques Protocols

1. Introduction

During the last years, a number of authenticated group key agreement protocols have been proposed in the literature. We observed that the efforts in this domain were mostly dedicated to the improvement of their performance in terms of bandwidth or computational requirements, but that there were very few systematic studies on their security properties. In this chapter, we propose a systematic way to analyze protocol suites extending the Diffie-Hellman key-exchange scheme to a group setting and presented in the context of the Cliques project. This leads us to propose a very simple machinery that will allow us to manually pinpoint several unpublished attacks against the main security properties claimed in the definition of these protocols (implicit key agreement, perfect forward secrecy, resistance to known-key attacks).

We start this chapter with an introduction to the A-GDH.2 protocol and the way it has been constructed from classical cryptographic protocols. We then describe (Section 3) security properties of interest in the context of AGKAP protocols. In Section 4, we construct a model allowing us to capture a number of properties of the A-GDH.2 protocol (and those of the other authenticated Cliques protocols), and propose a method for checking the security of these protocols. We successively consider the A-GDH.2 protocol itself (Section 5), its use in parallel with the A-GDH.2-MA protocol (Section 6) and the SA-GDH.2 protocol (Section 7), which is intended to provide stronger authentication properties than the A-GDH.2 protocol.

2. The A-GDH.2 Protocol

An Authenticated Group Key Agreement Protocol (AGKAP for short) is a protocol enabling a group of users to generate a shared secret key on a network that might be controlled by an active attacker.

Key agreement is one of the fundamental problems considered in cryptography. Probably the most well-known key agreement protocol is the Diffie-Hellman key agreement protocol [23]. A typical run of this protocol is represented in Fig. 1.1.

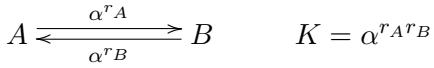


FIGURE 1.1. Diffie-Hellman Key Agreement protocol

In this figure, Alice and Bob want to share a session key K. They initially agree upon a multiplicative group $\mathcal{G}$ of prime order q and on a generator α. These values are public. Then, Alice generates a secret random value r_A while Bob does the same and generates r_B. In the next step, both users exponentiate the group generator α with their random value and exchange the result as represented in Fig. 1.1. At that moment, they may exponentiate the value they received with the random value they previously generated, and both will obtain $\alpha^{r_A r_B}$ which they decide to be the key.

The security of this protocol is based on a reputed hard problem: the Decisional Diffie-Hellman problem.

Definition 1.1 Decisional Diffie-Hellman (DDH) Problem: *Given a multiplicative group $\mathcal{G}$, a generator α of $\mathcal{G}$ and three elements of $\mathcal{G}$ $g_1 = \alpha^a$, $g_2 = \alpha^b$ and $g_3 = \alpha^c$, decide whether $c = a \cdot b$ or not.*

If the (passive) intruder is not able to solve this problem with a probability sensibly higher than $\frac{1}{2}$, then he is also unable to distinguish the key K from a random value.

In this thesis, we consider extensions of this protocol into two directions. First, we would like to extend this two-party protocol in order to enable a pool of users to share a key. We will also require the constitution of this pool to be dynamically modifiable at a limited cost. Furthermore, we would like to add authentication properties.

Among the different extensions of this protocol to a group setting, one of them retained our attention: the protocols proposed by Ateniese & al. in [6, 7]. A first interesting point was that these papers were based on extensions of the Diffie-Hellman protocol proved to be secure against a passive adversary [73] (their reduction has been improved in [16]). A second interesting point was the great regularity and simplicity of definition of these protocols: even when authentication was considered a goal, the only computation performed by the participants remained exponentiation. The authors did not desired to exploit signature or hashing primitives because they wanted the security of their protocols to rely on only one well-known computational problem.

One of their basic group key agreement protocols (without authentication) was defined as follows [73]:

Protocol 1 : GDH.2 Protocol

This protocol allows a group M of n users $M_1, \ldots, M_n$ arranged into a ring to share a key. Without loss of generality, we may assume that this arrangement corresponds to the lexicographic order. We assume p to be a prime integer and q a prime divisor of $p-1$. $\mathcal{G}$ is the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order q, and α is a generator of $\mathcal{G}$. $\mathcal{G}$ and α are public. Each group member M_i is assumed to select a new secret random value $r_i \in \mathbb{Z}_q^*$ during each session of the protocol. The messages are exchanged as follows:

Round i $(1 \leq i < n)$:

$M_i \rightarrow M_{i+1}$: $\quad \{\alpha^{\frac{r_1 \ldots r_i}{r_j}} | j \in [1,i]\}$, $\alpha^{r_1 \ldots r_i}$

Round n:

$M_n \rightarrow$ All M_i: $\quad \{\alpha^{\frac{r_1 \ldots r_n}{r_i}} | i \in [1,n[\}$

Upon receipt of the above, every M_i computes the group key as:

$$S_n \quad = \quad \alpha^{\frac{r_1 \ldots r_n}{r_i} . r_i} = \alpha^{r_1 \ldots r_n}$$

The messages exchanged during a GDH.2 protocol run with four participants are represented in Fig. 1.2.

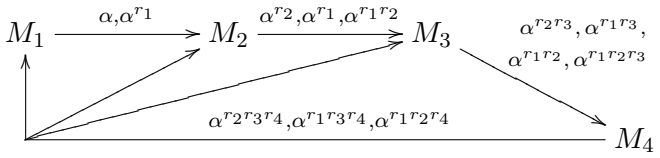


FIGURE 1.2. GDH.2 Protocol Run with 4 Participants

So, at this point, we have a group key agreement protocol. We will now see how Ateniese & al. added authentication properties to this protocol.

The fundamental problem is that the two-parties Diffie-Hellman protocol does not provide authentication services, what may result in the classical man-in-the-middle attack represented in Fig. 1.3.

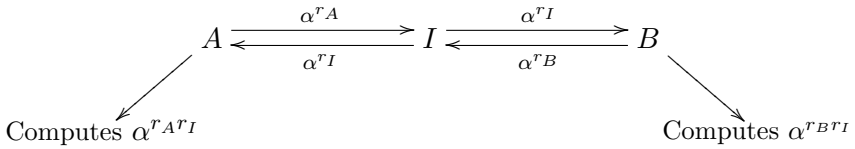


FIGURE 1.3. Lack of authentication in the D-H Protocol

In this scenario, A and B are trying to exchange a key, but the intruder, I, intercepts their messages and replaces them with a value α^{r_I} he generated himself. So, finally, A and B are both sharing a key with the intruder rather than with each other.

Ateniese & al. solve this problem as follows. They assume that each pair of users (M_i, M_j) possesses a secret long-term key K_{ij} in common. This may for instance be realized by supposing that each user M_i has a long-term public key α^{x_i} while the corresponding private key is x_i. In this setting, M_i and M_j may define the long-term secret key they are sharing as $K_{ij} = f(\alpha^{x_i x_j})$ where f is a mapping from $\mathcal{G}$ to $\mathbb{Z}_q^*$.

Starting from this assumption, they modify the flows of the two parties Diffie-Hellman key agreement as follows:

$$A \underset{\alpha^{r_B} K_{AB}}{\overset{\alpha^{r_A}}{\rightleftarrows}} B \qquad K = \alpha^{r_A r_B}$$

FIGURE 1.4. A-DH Key Agreement protocol

This protocol is the same as the original Diffie-Hellman protocol, except that B exponentiates α with $r_B K_{AB}$ rather than simply with r_B and A computes its key by exponentiating the message he receives with $r_A K_{AB}^{-1}$ rather than simply with r_A. The session key remains $\alpha^{r_A r_B}$.

The authors give informal arguments to "prove" the key authentication property, and the methods we are presenting hereunder do not allowed us to attack the key authentication property for this protocol.

Having defined this protocol, Ateniese & al. present the A-GDH.2 protocol that is a natural extension of the A-DH and GDH.2 protocols.

Protocol 2 : A-GDH.2 Protocol

This protocol allows a group M of n users $M_1, \ldots, M_n$ arranged into a ring to share a key. We assume α and $\mathcal{G}$ to be defined as in the GDH.2 protocol. Each group member M_i is assumed to select a secret random value $r_i \in \mathbb{Z}_q^*$ and each pair of users (M_i, M_j) is assumed to share a long-term secret key K_{ij}. The messages are exchanged as follows:

Round i $(1 \leq i < n)$:

$M_i \rightarrow M_{i+1}$: $\{\alpha^{\frac{r_1 \ldots r_i}{r_j}} | j \in [1, i]\}$, $\alpha^{r_1 \ldots r_i}$

Round n:

$M_n \rightarrow$ All M_i: $\{\alpha^{\frac{r_1 \ldots r_n}{r_i}} K_{in} | i \in [1, n[\}$

Upon receipt of the above, every M_i computes the group key as:

$$S_n \quad = \quad \alpha^{\frac{r_1 \ldots r_n}{r_i} \cdot r_i \cdot K_{in}^{-1}} = \alpha^{r_1 \ldots r_n}$$

A typical run of this protocol with 4 participants is represented in Fig. 1.5.

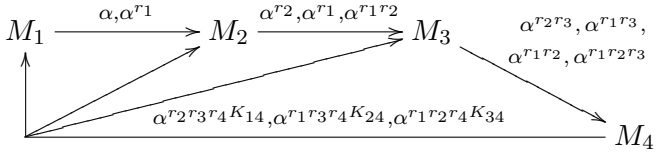


FIGURE 1.5. A-GDH.2 Protocol Run with 4 Participants

This protocol is the first one we will examine. We now discuss the security properties of interest in the context of these protocols.

3. Intended Security Properties

AGKAP are intended to be executed in the presence of an active attacker. This means that their security properties have to be valid in the presence of an attacker that is able to eavesdrop or intercept all the transmitted messages, as well as to send messages he generated using the information he possesses.

Starting from considerations given in [7], we suggest several security properties AGKAP should achieve to be exploitable in the practice.

3.1. Implicit Key Authentication

In the context of AGKAP, this property may be defined as follows.

Definition 1.2 *A protocol is said to achieve* Implicit Key Authentication *(IKA) if, when he completed his role in a session of the protocol, each $M_i \in \mathsf{M}$ is assured that no party $M_I \notin \mathsf{M}$ can learn the key $S_n(M_i)$ (i.e. M_i's view of the key).*

This property does not mean that all group members have any knowledge of a group key at the end of the protocol, nor that they agree on its value. Also, it does not imply for a group member that any other member executed a session of the protocol (there is no liveness property intended in the sense of G. Lowe in [46]).

3.2. Perfect Forward Secrecy

This property is defined in [53] as follows:

Definition 1.3 *A protocol is said to achieve* Perfect Forward Secrecy *if compromise of long-term keys does not compromise past session keys.*

Traditionally used in a two parties setting, this property is exploited in a context where no message was manipulated in the past. This is natural since the two users cannot exchange messages using the session key if they do not agree on its value. However, things are slightly different in a multiparty setting: the manipulation of a message can alter the view of the key of one particular user, while all other group members are computing the same key; and this manipulation does not prevent these last users from communicating.

So, we propose to define two flavors of perfect forward secrecy: *complete forward secrecy* and *individual forward secrecy*. According to the former, no message has been manipulated in the past for any member of the group; but according to the latter, one (or a few) member(s) of the group may have been subject to attacks which left the other members of the group unaffected (and thus the protocol was individually correct for each of them). This scenario is plausible since, after executing the protocol, the attacked individual(s) may just be passive recipient(s) of the messages exchanged by the others. We summarize this in the two following definitions.

Definition 1.4 *A protocol is said to achieve* Complete Forward Secrecy *if compromise of long-term keys does not compromise past session keys, assuming that no message of the previous sessions of the protocol has been manipulated.*

Definition 1.5 *A protocol is said to achieve* Individual Forward Secrecy *if compromise of long-term keys does not compromise past session keys, assuming that some group members may have been subject to attacks in the past, leaving the other group members unaffected.*

3.3. Resistance to Known-Key Attacks

This property is defined in [53] as follows:

Definition 1.6 *A protocol is said to be vulnerable to a* known-key attack *if compromise of past session keys allows either a passive adversary to compromise future session keys, or impersonation by an active adversary in the future.*

One of the main motivations for the definition of this property is the protection of future sessions against the compromise of session secrets usually more weakly protected than long-term ones.

However, in the protocols we analyze, the session keys are not the only session secrets: each user generates a contribution to the session key that is also kept secret. So, we think it is judicious to consider whether security problems can result in further sessions of the compromise of any secret local to past sessions (rather than only of the compromise of session keys). We then introduce the following property:

Definition 1.7 *A protocol is said to be vulnerable to a* known session-secret attack *if compromise of past session secrets allows either a passive adversary to compromise future session keys, or impersonation by an active adversary in the future.*

In the case of the A-GDH.2 protocol, the verification of this property would lead us to consider the consequences of the compromise of some past random contribution r_i on future sessions. It may be observed that a protocol that is subject to known-key attacks is trivially also subject to known session-secrets attacks.

The A-GDH.2 protocol is claimed to achieve implicit key authentication and complete forward secrecy. It is however shown to be subject to an unpractical known-key attack.

4. A Model for the Analysis of the Cliques Protocols

Having described a typical Cliques protocol and the properties we will have to check for AGKAP in general, we now build a model that we will use for the analysis of these protocols.

We first describe some related works and the basic motivations for our model. Then, we describe the way we represent the exchanged messages and the intruder capabilities and, finally, we show how the intended security properties can be verified.

4.1. Introduction and Related Works

Although the experience has shown how complex it is to define security protocols that can be used in the presence of active attackers (see [20, 45, 26] for instance), AGKAP's have rarely been systematically studied until recently: only sketch proofs or informal arguments were given to convince of their correctness in their presentation [7, 19].

Two types of models, coming from two little related communities, have been developed for the study of security protocols. Computational approaches have been proposed: we can notably mention the work of Bellare and Rogaway [9] that has been extended by Blake-Wilson, Johnson and Menezes [10] in order to enable the handling of the authenticated Diffie-Hellman key exchange. More recently, Bresson & al. [13, 14], Steiner [72] and Katz and Yung [40] further extended these methods for the analysis of AGKAP's. In these approaches, cryptographic operations are considered as functions on bit strings and security properties are expressed in terms of probability and computational complexity of successful attacks. Unfortunately, proofs using such methods are often laborious and do not render pointless the development of analysis methods for reasoning at a higher level of abstraction. In these methods, security properties are formally (logically) modelled and cryptographic

operations are viewed as functions on a space of symbolic expressions. These analyzes allowed researchers to capture a lot of useful intuitions about security protocols and discover many new flaws by considering only idealized cryptographic primitives and without precisely taking the computational capabilities of the intruder into account. The methods we are using in this chapter can be placed in the second category although they capture arithmetic properties at a lower level of abstraction than the one usually considered in this type of methods.

A number of methods were developed during the last few years for the formal (or logical) analysis of security protocols. Many of them are based on state-space exploration: they usually proceed by defining an arbitrarily bounded system and exploring it, hoping (or after having proved) that, if there is an error in the protocol, it can be described by a behavior included in the considered state-space [27, 47, 48, 58]. However several tools allow proofs to be obtained for unbounded systems at the cost of the interactive proof of several lemmas [49] or at the risk of receiving no answer for some protocols [35, 70].

Other approaches are based on the use of logics [61, 76]. They allow proofs to be obtained for arbitrary size configurations, but they often require particularly error-prone and time-consuming formalization steps and do not provide the same support in pinpointing problems as the direct generation of counter examples.

Recently, "manual" approaches were also presented, allowing fine-grained proofs to be obtained for systems containing an unbounded number of occurrences of each defined roles [36, 78].

The use of state-space exploration techniques in the study of group protocols seems very difficult due to their very essence: the number of participants in a honest session of the protocol is basically unbounded. This will intuitively result in dramatic state-space explosion problems. As far as we know, the only successful analyzes of group protocols have been performed by theorem-proving approaches [18, 60], which allow inductive reasoning. Besides this, C. Meadows has performed (independently of us) the analysis of the A-GDH.2 protocol, adapting her NRL Protocol Analyzer by extending the power and scope of its theorem-proving capabilities [50]. In this work, she proved that a group key cannot be compromised, provided that the considered group never contains any dishonest members [52].

Beyond the problem of the unbounded number of participants in the protocols, the modelling of the A-GDH.2 protocols suite requires the capture of several low-level arithmetic properties: exponentiation, commutativity, associativity that are out of the scope of most of the works encountered in the literature. Furthermore, the A-GDH.2 key generation protocol is not intended to be used alone: there are several

other protocols in the suite (member addition, deletion, group merging, ...) that use values computed during the key generation protocol and can interfere with its security properties.

In the next paragraphs of this section, we introduce the way we model the exchanged messages, describe the intruder capabilities and, finally, we show how the intended security properties can be verified.

4.2. Messages and Intruder's Knowledge

The messages sent during the execution of the protocols we are analyzing are constituted by the concatenation of elements of a group $\mathcal{G}$ of prime order q. A particular element, that we denote α, is a generator of $\mathcal{G}$ and is shared by all users of the network (as well as the knowledge of the characteristics of the group $\mathcal{G}$). All exchanged elements of $\mathcal{G}$ are expressed as powers of α. It can then be checked that the participants have to manipulate three types of elements:

- Random numbers (r_i)
- Long-term keys (K_{ij})
- Elements of $\mathcal{G}$ expressed as α raised to the power of a product of random numbers and long-term keys.

The behavior of the honest participants is quite simple: they receive elements of $\mathcal{G}$, exponentiate them with random numbers and/or long-term keys (possibly inverted), and send the result to other participants. The group-key is obtained in the same manner, except that the result of the computations is not sent but kept secret. It can be noticed that, when a participant receives an element of $\mathcal{G}$, he has to accept it without being able to check anything concerning its constitution or origin. The goal of an intruder will therefore be to possess a pair (α^x, α^y) of elements of $\mathcal{G}$ related between them in such a way that α^y is equal to the result of the key-computation operation of a honest M_i applied to α^x. If we take this point of view, there are n secret pairs corresponding to an execution of the protocol between n parties.

Example 1.1 If we refer to Fig. 1.5, the goal of the intruder who wants to fool M_1 is to obtain a pair $(\alpha^x, \alpha^{xr_1K_{14}^{-1}})$ for some x: if he obtains this pair, he can replace the term $\alpha^{r_2r_3r_4K_{14}}$ with α^x and M_1 will compute $\alpha^{xr_1K_{14}^{-1}}$ as group key.

Since all properties of the protocols we are analyzing can be described in terms of relation between elements of $\mathcal{G}$, our model will not deal with elements of $\mathcal{G}$ but with the possible relations between pairs of them: the pair (α^x, α^y) will be modelled as the ratio y/x.

More precisely, our model considers the manipulation of two sets of elements:

- The set E containing the random numbers r_i and the long-term keys K_{ij}
- The set $\mathsf{P} = \{\prod r_i^{e_i} \prod K_{jl}^{e_{jl}} | e_i, e_{jl} \in \mathbb{Z}\}$ of all possible products of elements of E. This set will be used to represent the relation between elements of $\mathcal{G}$: $p \in \mathsf{P}$ will represent the pairs of elements of $\mathcal{G}$ that can be written as (α^x, α^{px}) for any value of x.

Example 1.2 With our notations, the secret pair corresponding to the secret of M_1 in Fig. 1.5 will be represented by the element $\frac{xr_1K_{14}^{-1}}{x} = r_1K_{14}^{-1} \in \mathsf{P}$.

The use of such a construction will be quite convenient and can be intuitively justified if we state an hypothesis that is quite similar to the widely used *perfect encryption assumption* [47, 48, 61, 78]. We will refer to this hypothesis as the *perfect Diffie-Hellman assumption* and it can be stated as follows:

Definition 1.8 Perfect Diffie-Hellman Assumption

An element of $\mathcal{G}$ can be computed in one and only way: by exponentiating the generator α with the correct random numbers and keys (excepted the permutations in the order of the exponentiation of α and the possibility of exponentiation by an element of E and by its inverse successively).

This assumption implies in particular that a secret element of $\mathcal{G}$ cannot be computed by combining other elements of $\mathcal{G}$ (but only elements of E with elements of $\mathcal{G}$). It seems quite plausible in practice since arithmetic is performed in a large group (lucky guesses or collisions are very unlikely) and since the DDH problem is hard.

It can also be noticed that the use of the P-set implies another restriction due to its very structure: it does not allow capturing relations between more than two elements of $\mathcal{G}$. However, in the presence of our perfect Diffie-Hellman assumption, the relevant security properties always come down to the impossibility of finding two elements of $\mathcal{G}$ presenting between them a particular relation, so that the consideration of more complex relations cannot be of any help to prove the correctness of Cliques protocols. It could be useful to use such extensions to discover more dangerous attacks that violate more than one security property, but we are more interested in checking the correctness of protocols than in finding "optimal" attack sketches.

We will now be looking at the ways the intruder can use to manipulate our two sets of elements.

4.3. Intruder Capabilities

Considering our perfect Diffie-Hellman assumption, the only useful computation for the intruder will be the exponentiation of an element of $\mathcal{G}$ with a known element of E. If we note E_I and P_I the subsets of E and P (respectively) that are known by the intruder, we can then transpose this remark as follows:

$$\begin{array}{c}\text{(1) If } e \in \mathsf{E}_I \text{ and } p \in \mathsf{P}_I \text{ then} \\ pe \in \mathsf{P}_I \text{ and } pe^{-1} \in \mathsf{P}_I\end{array}$$

Example 1.3 Consider the intruder knows a pair $(\alpha^{r_1}, \alpha^{r_1 r_2})$ and some random value r_I. Then $r_1 r_2 / r_1 = r_2 \in \mathsf{P}_I$ and $r_I \in \mathsf{E}_I$. Since the intruder is able to exponentiate any element of $\mathcal{G}$ with r_I, he is able to compute the two following pairs: $(\alpha^{r_1 r_I}, \alpha^{r_1 r_2})$ and $(\alpha^{r_1}, \alpha^{r_1 r_2 r_I})$, which correspond to $r_2 r_I^{-1} \in \mathsf{P}_I$ and $r_2 r_I \in \mathsf{P}_I$ respectively.

There is another way for the intruder to obtain new elements of P: the use of the computations executed by the honest users. As we said above, the behavior of these users is quite simple: they receive elements of $\mathcal{G}$ and exponentiate them with some values of E or, sometimes, they provide some elements of $\mathcal{G}$ directly computed from the public generator α. We will call such operations *services*. More precisely, a *service* is a function $s : \mathcal{G} \to \mathcal{G} : \alpha^x \to \alpha^{p.x}$, and we call S the set of available services. For simplicity purposes, we will refer to the service $s(\alpha^x) \to \alpha^{xp}$ by the power it raises its input: $p \in \mathsf{S}$. Let us see how a service can be described in term of growth of P_I. If $p \in \mathsf{P}_I$, then the intruder possesses two elements of $\mathcal{G}$ that can be written α^x and α^{px}. So, if the intruder sends α^x to a honest user performing the service s, then he will learn the element $s^{-1}p \in \mathsf{P}_I$. Conversely, if the intruder sends α^{px} to the user performing the same service, he will learn the element $sp \in \mathsf{P}_I$. We can then write our second rule for the increasing of the P_I-set:

$$\begin{array}{c}\text{(2) If } s \in \mathsf{S}, \text{ and } p \in \mathsf{P}_I \text{ then} \\ sp \in \mathsf{P}_I \text{ and } s^{-1}p \in \mathsf{P}_I\end{array}$$

Example 1.4 Consider that the intruder possesses the pair $(\alpha^{r_1}, \alpha^{r_1 r_2})$ and that, during the execution of some AGKAP, the user M_3 exponentiates values he receives with r_3 and sends the result to another group member. Then $r_2 \in \mathsf{P}_I$ and $r_3 \in \mathsf{S}$. If the intruder replaces the value M_3 received as input for the r_3 service with α^{r_1}, then M_3 will send $\alpha^{r_1 r_3}$ and the intruder comes in possession of a pair $(\alpha^{r_1 r_3}, \alpha^{r_1 r_2})$ to which corresponds $r_2 r_3^{-1} \in \mathsf{P}_I$. Conversely, if the intruder replaces the input of the service r_3 with $\alpha^{r_1 r_2}$, then M_3 will send $\alpha^{r_1 r_2 r_3}$ and the intruder comes in possession of a pair $(\alpha^{r_1}, \alpha^{r_1 r_2 r_3})$ to which corresponds $r_2 r_3 \in \mathsf{P}_I$.

We have however to be careful in the use of this rule and will have to impose some restrictions in its application (due to the fact that honest users are providing services the input of which is fixed to α for instance). This will be examined more in the details in the next section where we will propose a method to determine whether a ratio is secret or can be obtained by the intruder.

4.4. Proving Security Properties

In this section, we suggest a systematic process to obtain the proof of the secrecy of a particular $p \in \mathsf{P}$ in a selected system, i.e. to prove that some elements of P cannot be obtained from the initial knowledge of the intruder by using the two rules we defined in the previous section.

4.4.1. Definition of the system

At first, we define the system we will analyze: we fix the sessions of the protocol we are considering and the intended roles for each user. This allows us to define the available services (S-set), the atomic elements initially known by the intruder (E_I-set), and the secret ratios (let P_S be this set).

Example 1.5 Let us consider the system containing the session represented in Fig. 1.5 and assume that r_3 is compromised (because of the compromising of the pseudo-random generator of M_3 for example).

During the first three rounds, the user M_i provides the service $r_i \in \mathsf{S}$ several times in parallel. During the fourth round, M_4 provides 3 services: r_4K_{14}, r_4K_{24} and r_4K_{34}. The secret of M_i $(1 \leq i < 4)$ is $r_iK_{i4}^{-1}$ while the secret of M_4 is r_4. So, to summarize, the sets we will consider are the following:

$$\begin{aligned} \mathsf{S} &= \{r_1, r_2, r_3, r_4K_{14}, r_4K_{24}, r_4K_{34}\} \\ \mathsf{E}_I &= \{r_3\} \\ \mathsf{P}_S &= \{r_1K_{14}^{-1}, r_2K_{24}^{-1}, r_3K_{34}^{-1}, r_4\} \end{aligned}$$

In the definition of the A-GDH.2 protocol, an occurrence of the r_2 service can be interpreted in two ways: α^{r_2} may be computed from the term M_2 receives from M_1 or from the publicly known group generator α. We will assume in the rest of this paper that α^{r_2} is computed by exponentiating the term M_2 received as first element of the first message of the protocol (without checking the correctness of this value) rather than by generating it from the public group generator. Assuming this, the last way of executing the protocol should only change minor details in the further developments.

4.4.2. Use of the (1)-rule

All elements corresponding to those of E_I can be deleted from the expression of S and P_S. This operation simplifies our problem and does not change its solutions since:

- If $e \in \mathsf{E}_I$, each operation that uses the service $e^a s \in \mathsf{S}$ ($a \in \mathbb{Z}$) can be performed by using a service s and by suitably applying the (1)-rule.
- If $e \in \mathsf{E}_I$, and $p.e^a \in \mathsf{P}_S$ ($a \in \mathbb{Z}$) then the knowledge of p implies the one of $p.e^a$ (anew by applying (1)-rule).

We will call the resulting sets S^1 and P_S^1.

Example 1.6 Applying this step to the sets obtained in the previous example provides:

$$\begin{aligned} \mathsf{S}^1 &= \{r_1, r_2, r_4K_{14}, r_4K_{24}, r_4K_{34}\} \\ \mathsf{P}_S^1 &= \{r_1K_{14}^{-1}, r_2K_{24}^{-1}, K_{34}^{-1}, r_4\} \end{aligned}$$

We can observe that the service r_3 has completely disappeared. The reason is that this service is not useful anymore if r_3 is known by the intruder.

4.4.3. Use of the (2)-rule

The problem is now to determine whether some element of P_S^1 can be obtained from S^1 by suitably exploiting the (2)-rule.

The secrecy of a particular element of P_S can be determined by writing a linear system $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ expressing the "balance" of the variables in the construction of the secret from the services. This system contains one variable per element of S^1 and one equation per element in E. On the left side of the equation, the value of each element of the matrix $\mathbf{A}$ is the power of the element of E corresponding to the line in the element of S^1 corresponding to the column. The second term of each equation is the power of the element of E corresponding to the line in the studied secret.

This system expresses that the only way to compute the secret is to successively apply some services starting from the only initially known ratio: 1. If this system is inconsistent, then the intended confidentiality property is verified (in our model, and for the considered system).

Example 1.7 If we apply this last step on the previous example, the linear system corresponding to the verification of the secrecy of $r_1K_{14}^{-1}$ is:

$$\begin{array}{c} \\ r_1 \\ r_2 \\ r_4 \\ K_{14} \\ K_{24} \\ K_{34} \end{array} \begin{array}{c} \begin{array}{ccccc} r_1 & r_2 & r_4K_{14} & r_4K_{24} & r_4K_{34} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

It can be observed that this linear system has no solution since the last four equations are inconsistent. So, we are not able to obtain the secret $r_1 K_{14}^{-1}$ from the studied system.

4.4.4. Reconstruction of an attack

If the system resolved in the previous point presents some solutions, we have to check whether these solutions can lead us to an attack against the considered protocol: this is not always the case, due to some limitations that we have to introduce in the exploitation of the (2)-rule as we mentioned at the end of Section 4.3.

In the context of the Cliques protocols, the most general message transformation executed by a user during a single step of the protocol can be described as follows:

$$\alpha^{x_1} \dots \alpha^{x_n} \rightarrow \alpha^{y_1} \dots \alpha^{y_m} \alpha^{x_1 y_{11}} \alpha^{x_1 y_{12}} \dots \alpha^{x_n y_{n1}} \dots \alpha^{x_n y_{np}}$$

This expression means that a protocol participant will never receive anything else than a sequence of elements of $\mathcal{G}$, and will never send anything else than a similar sequence, which can contain two types of elements:

- elements of $\mathcal{G}$ generated by exponentiating the group generator α with a known value y_i;
- elements of $\mathcal{G}$ generated by exponentiating a just received element α^{x_i} with a known value y_{ij}.

In the expression above, the indexes m, n and p may be null, and the order of the sent values can naturally be permuted.

This assumption keeps out of our scope the more general case where a protocol participant may exponentiate any previously received value (and not only those he just received). We will however examine this possibility in the next chapter.

We now start from this assumption to define the rules limiting the composition of services in the derivation of the set P_I.

It is possible to build an attack against the user M_i if the intruder is able to exploit the services s_i specified by the solution of the linear system in order to obtain a pair of the form $(\alpha^x, \alpha^{xp_s})$ (where p_s is M_i's

secret) and if he is able to replace the value M_i will use to compute his view of the group key with α^x.

We first determine the conditions upon which the required pair may be obtained. Suppose that the linear system solved in the previous section shows that the secret p_s may be expressed as the following combination of services:

$$p_s = x_1 s_1 + \cdots + x_s s_s$$

In this equation, the "+" symbol denotes the composition of services, while the x_i coefficients are the solutions of the previous linear system and indicate how many times each service will have to be exploited in order to obtain a pair of the form $(\alpha^x, \alpha^{xp_s})$, a negative sign corresponding to the submission of the first element of the pair as entry to the service, while a positive sign corresponds to the submission of the second element of the pair as entry to the service (as described when we established the (2)-rule). We will therefore say that a service is used in the *negative* (resp. *positive*) *direction* when this service has to be provided on the first (resp. second) term of the pair.

It can however be observed that these submissions are not always feasible in the practice.

At first, we have to keep in mind that the group members are executing the protocol round by round, and that during each round they receive a sequence of elements of $\mathcal{G}$ and are sending another such sequence. So, it may be observed that two services provided during the same round cannot be exploited in the same direction: we would have to submit the output value of one of them as entry to the second, what is not possible anymore since the round is completed when we are in possession of the first output. This observation leads us to express a first necessary condition for the possibility of building a pair by exploiting a solution of a linear system:

Condition 1 When constructing a pair of elements of $\mathcal{G}$, we may exploit at most two services within one single round (and each of them only once), and these services must be exploited in opposite directions.

Besides, it is not always possible to submit a value as entry for a service: some of them are provided by group members computing values directly form the group generator α (it is the case for α^{r_1} in the run of the A-GDH.2 protocol represented in Fig. 1.5 for instance). These services may then only be used to provide starting values in the desired pair, and we therefore may use two services of this form (we will call them *starting services*) at most, one in the negative direction, and one in the positive direction.

There is another case leading to similar consequences: when, within a round, one wishes to exploit two services s_1 and s_2 using the same input value (we will refer to this as using a *splitting service*). In that case, whatever we submit as input value for these services, we will always obtain two elements corresponding to the ratios $s_1 s_2^{-1}$ or $s_1^{-1} s_2$ (according to the order in which we place the services outputs). We may observe that, as when exploiting starting services, the obtained ratio is independent of any value previously known, and we anew have to use splitting services before any other. This is expressed in the following condition:

Condition 2 When constructing a pair of elements of $\mathcal{G}$, we may exploit at most one splitting service and at most two starting services, provided that they are exploited in opposite directions. Furthermore, if we are using a splitting service, we may not use any starting service.

Furthermore, since these particular services have to be exploited at first when constructing some pair, we are not able to exploit any service a user would provide before them and in the same direction. We express this more precisely in the three following conditions:

Condition 3 If a splitting service provided by M_x is used when constructing a pair of elements of $\mathcal{G}$, then we cannot use any service M_x should provide before it.

Condition 4 If one starting service provided by M_x is used in a certain direction when constructing a pair of elements of $\mathcal{G}$, then we cannot use any service M_x should provide in the same direction before it.

Condition 5 If two starting services provided by M_x and M_y (it is possible that $M_x = M_y$) are used when constructing a pair of elements of $\mathcal{G}$, then we cannot use the services provided by both users before the considered starting services (but we can use those provided by only one of these users, provided that they are used in the direction opposite to the one of the starting service provided by that user)

If these constraints are verified, we are able to build the desired pair by using Algorithm 1 (for instance). In this (high-level) algorithm, we are using some **Collect** function that is described in Algorithm 2.

This algorithm may be justified as follows.

First, Algorithm 1 states that the intruder is isolating all group members. This is usually not necessary in the practice, but it allows us to consider the execution of the protocol member after member, in the order that we want, without needing to care about the interactions the users may have. This is possible due to the fact that all messages exchanged during some session of a Cliques protocol may be constructed

Algorithm 1 Provides a pair (g_1, g_2) the ratio of which is equal to a specified product of services

The intruder isolates all group members: he is intercepting all messages they are sending
$g_1 := \alpha \quad g_2 := \alpha$
if some splitting service offered by M_x has to be used **then**
  Collect the services offered by M_x
  Collect the other services member by member
else if two starting services s and s' offered by M_x and M_y are used **then**
  We choose that s, provided by M_x is a service not preceded by any other necessary services
  $g_1 :=$ Output of s if it is used in the negative direction
  $g_2 :=$ Output of s if it is used in the positive direction
  Collect the services provided by M_y
  Collect the remaining services member by member
else if only one starting service offered by M_x is used **then**
  Collect the services offered by M_x
  Collect the other services member by member
else
  Collect the services member by member
end if

Algorithm 2 Collect the services offered by some member M_x into the pair (g_1, g_2)

for $j := 1$ to the number of rounds executed by M_x **do**
  if some service s provided by M_x during the j-th round has to be used in the negative direction **then**
    g_1 is provided (if possible) as input for s and is updated with its output
  end if
  if some service s' provided by M_x during the j-th round has to be used in the positive direction **then**
    g_2 is provided (if possible) as input for s' and is updated with its output
  end if
  Dummy values are given for the unaffected inputs expected in the considered round
end for

(from the structural point of view) by anyone, the intruder included: the only condition for a message to be valid is its length, publicly specified in the protocol definition.

This is normally not a security problem since, in the Cliques protocols, a user may complete a session of the protocol, apparently executed with $n-1$ other group members, without these members having sent or received any message. As we said above, there is no liveness property for this protocol family.

Then, Algorithm 1 affects initial values to g_1 and g_2 that will form the desired pair at the end of its execution. At that point, we start the **if** clauses describing the different steps to be performed according to the use of splitting or starting services. As imposed by condition 2, there are four possible cases: one splitting service is used, two starting services are used, one starting service is used (and no splitting service is used in these last two cases) or no splitting nor starting service is used.

We consider the case where two starting services are used: the other cases are very similar to that one.

From Condition 5, at most one of these starting services is offered as the first necessary service provided by a member. We say that it is the service s offered by M_x. Then, we affect the output of s to g_1 or g_2 according to the direction s has to be used. We suppose this direction to be negative, so g_1 has been affected (the other case being completely symmetric).

Then, we collect the services M_y provides by applying Algorithm 2. This algorithm proceeds by going through the rounds executed by M_y in the order they are offered. For each round, the intruder checks whether it contains services that have to be exploited to build the desired pair (by Condition 1, there is at most one service in each direction that may be used during each round), and provides g_1 or g_2 as input for these services according to the direction in which they have to be exploited (when such an input exists), the other values of the round being affected randomly. M_y executes the round, and g_1 and g_2 are updated with the output of the considered services.

If we look at the rounds executed by M_y, they may be divided into two categories: those provided before (or during the same round as) s', and the others. The services of the first category must be offered in the same direction as s (Conditions 1, 2, 4 and 5). For that reason, when applying Algorithm 2 for M_y, only g_1 may be changed during the rounds preceding the one where s' is executed, so what would be the input of s' remains α and g_2 is affected to the output of s'.

Finally, we may use Algorithm 2 for the remaining rounds executed by M_x and for the $n-2$ other group members.

So, at that point, we are able to obtain a pair (g_1, g_2) of elements of $\mathcal{G}$ that can be used to attack M_i : $g_2 = g_1^{p_s}$. We however still have to check one thing: we need to be able to send the left-term of the obtained

pair as the value that M_i will be using to compute his view of the group key. This will impose us two constraints more. First:

Condition 6 When attacking M_i, we cannot use in the negative direction any service he provides during the round from which he computes his view of the group key, nor after that round.

This is necessary since we need to be in possession of the first term of our pair (i.e. g_1) before M_i executes the round from which he will compute his view of the group key.

Our sixth condition only concerns services used in the negative direction. There is however another problem which may occur: we may need to use in the positive direction a service whose input is the value that M_i will use to compute his view of the group key. This is problematic since it would impose us to submit g_2 as input for this service rather than g_1. This problem is prevented if the following condition is respected:

Condition 7 When attacking M_i, we cannot use any service the input of which is the element of $\mathcal{G}$ that M_i uses to compute his view of the group key.

The use of these seven conditions and of the two algorithms above is exemplified in Appendix B for the building of an attack described in Section 5.1.2.

We will now see how our analysis method can be applied for the analysis of the A-GDH.2 protocols suite.

5. Analysis of the A-GDH.2 Protocol

In the following paragraphs, we will analyze the security properties of the A-GDH.2 protocol described in Section 2. The A-GDH.2 protocol was claimed to provide implicit authentication and perfect forward secrecy. However, a very impractical known-key attack was suggested [7].

5.1. Implicit Key Authentication

We will divide our analysis into two parts: in the first one we will assume that the intruder is not a member of any group, while in the other we will consider that the intruder is a legitimate member of some groups.

5.1.1. IKA when the intruder is excluded from all groups

As described in the previous section, the first step in our analysis will be the description of the protocol.

Initially, we will consider only one session of the protocol with n participants. The intruder knowing no long-term key nor any short-term secrets, the E_I-set is to be empty. From the first to the $(n-1)$-th round, the user M_i provides the service $r_i \in \mathsf{S}$ several times in parallel. During the n-th round, M_n provides the $n-1$ services: $r_n K_{1n}$, ..., $r_n K_{n-1n}$. The secrets are the following: $r_i K_{in}^{-1}$ for M_i $(1 \leq i < n)$ and r_n for M_n. So, to summarize, the sets we consider are the following:

$$\begin{aligned} \mathsf{S} &= \{r_1, r_2, \ldots, r_{n-1}, r_n K_{1n}, \ldots, r_n K_{n-1n}\} \\ \mathsf{E}_I &= \emptyset \\ \mathsf{P}_S &= \{r_1 K_{1n}^{-1}, \ldots, r_{n-1} K_{n-1n}^{-1}, r_n\} \end{aligned}$$

If we follow the analysis scheme proposed above, we now have to express the linear system describing the "balance" of the variables of E to check the secrecy of the elements of P_S. We first look at the secrecy of $r_1 K_{1n}^{-1}$. If we use the "s"-letter to denote the variable indicating how many times the service s has to be used to construct the secret, the linear system can be written as follows:

$r_1 = 1$	Balance on r_1
$r_i = 0$	Balance on r_i $(2 \leq i < n)$
$\sum_{i=1}^{n-1} r_n K_{in} = 0$	Balance on r_n
$r_n K_{1n} = -1$	Balance on K_{1n}
$r_n K_{in} = 0$	Balance on K_{in} $(2 \leq i < n)$

It can be observed that summing the $n-1$ equations corresponding to the balance on the keys provides an inconsistency with the equation expressing the balance on r_n. Hence we can say that $r_1 K_{1n}^{-1}$ cannot be obtained by using the two enrichment rules we defined and $S_n(M_1)$ is kept secret for this configuration as claimed in the protocol definition. If we write this system in the case of multiple sessions of the protocol (from which the intruder is excluded), it can easily be checked that this inconsistency is preserved. The transposition of this result for the $r_i K_{in}^{-1}$ -secrets is straightforward and if we transform the second members of these equations in order to prove the secrecy of r_n, we can easily obtain an inconsistency between the same equations. The consideration of several sessions of the protocol from which the intruder is excluded leads to the same results. We can then say that the Implicit Key Authentication property is correct (with respect to the assumptions of our model) provided that the intruder is not a member of any group.

5.1.2. IKA when the intruder is a legitimate member of some groups

We will now check whether this property is preserved when the intruder is a member of some groups. As a simple scenario, we will assume a first session of the protocol in which M_1, M_2, M_I and M_3 are the intended participants (M_3 being the group controller), and a second session with the same participants excepted M_I. We will note r'_i the random contribution generated by M_i during this last session. So, the sets of interest become:

$$\begin{aligned} \mathsf{S} &= \{r_1, r_2, r_I, r_3K_{13}, r_3K_{23}, r_3K_{I3}, \\ &\quad r'_1, r'_2, r'_3K_{13}, r'_3K_{23}\} \\ \mathsf{E}_I &= \{r_I, K_{I3}\} \\ \mathsf{P}_S &= \{r'_1K_{13}^{-1}, r'_2K_{23}^{-1}, r'_3\} \end{aligned}$$

The application of the second step of our proof scheme provides the following sets:

$$\begin{aligned} \mathsf{S}^1 &= \{r_1, r_2, r_3K_{13}, r_3K_{23}, r_3, \\ &\quad r'_1, r'_2, r'_3K_{13}, r'_3K_{23}\} \\ \mathsf{P}^1_S &= \{r'_1K_{13}^{-1}, r'_2K_{23}^{-1}, r'_3\} \end{aligned}$$

If we solve the three corresponding linear systems (i.e. one system per element of P_S), a number of solutions can be found.

We are choosing to verify the secrecy of $r'_2K_{23}^{-1}$ for instance. The linear system corresponding to this secret is the following one:

$$\begin{aligned} r_1 = r'_1 = r_2 = 0 \qquad & r'_2 = 1 \\ r_3K_{13} + r_3K_{23} + r_3 = 0 \qquad & r'_3K_{13} + r'_3K_{23} = 0 \\ r_3K_{13} + r'_3K_{13} = 0 \qquad & r_3K_{23} + r'_3K_{23} = -1 \end{aligned}$$

A solution to this linear system is:

$$r'_2 = 1 \qquad r_3K_{23} = -1 \qquad r_3 = 1$$

while all other variables are null [1].

It can be checked that all conditions defined in Section 4.4.4 are respected for this solution, so an attack can be constructed. We could use the algorithms provided in that section to build this attack, but we proceed here more heuristically, what will lead us to more intuitive attack scenarios. A systematic illustration of the considerations of Section 4.4.4 is provided in Appendix B.

To build this solution, we firstly have to exploit two services of the first session: r_3K_{23} and r_3K_{I3} (this last service is the one from which the r_3-term of the solution comes, the K_{I3} variable having been deleted

[1] $r'_2 = 1, r'_3K_{13} = 1, r'_3K_{23} = -1, r_3K_{13} = -1, r_3 = 1$ and all other variables set to 0 is another solution of the same system.

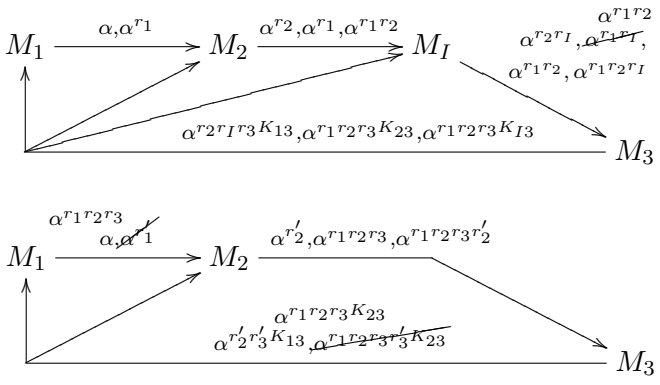
during the definition of the S^1-set). These services are provided by M_3 when he forms the final broadcast, and we have to exploit them in opposite directions (since the value of the variables indicating how they have to be used is 1 and -1 in the solution).

So, the intruder will choose an element of $\mathcal{G}$, say α^x, and place it in the message that M_3 receives in such a way that M_3 will exponentiate α^x with r_3K_{23} and r_3K_{I3}, providing the values $\alpha^{xr_3K_{23}}$ and $\alpha^{xr_3K_{I3}}$. The last term can be exponentiated by the intruder in order to obtain the pair $(\alpha^{xr_3K_{23}}, \alpha^{xr_3})$ that corresponds to the ratio $K_{23}^{-1} \in \mathsf{P}_I$.

During the second session, the intruder has to use the r'_2 service in the positive direction. This service is provided by M_2 during the second round, so the intruder will replace an element that M_2 receives from the first round with α^{xr_3} and M_2 will send $\alpha^{xr_3r'_2}$.

Finally, the intruder possesses a pair $(\alpha^{xr_3K_{23}}, \alpha^{xr_3r'_2})$ that corresponds to the secret of M_2 during the second session. So, if he replaces the term of the broadcast intended to M_2 with $\alpha^{xr_3K_{23}}$, M_2 will compute $\alpha^{xr_3r'_2}$ as the (secret) group key; and this term is known by the intruder.

An instantiation of this scenario where M_I has chosen α^x to be equal to $\alpha^{r_1r_2}$ is represented in Fig. 1.6.



$$S_3(M_2) = \alpha^{r_1r_2r_3r'_2}$$

FIGURE 1.6. Attack against Implicit Key Authentication

The key computed by the group controller M_3 can be obtained by using the following solution of the corresponding linear system:

$$r'_3K_{23} = 1 \qquad r_3K_{23} = -1 \qquad r_3 = 1$$

We can observe that giving the second place in the first session to M_I (instead of the third) does not change anything to the equations we solved. In fact, this would only change the number of times services

are provided in parallel, what does not affect the secrecy of particular elements in the case of the A-GDH.2 protocol. Moreover, the solutions we described remain valid when considering larger groups.

It can also be observed that a similar scenario can be applied in parallel against all members of the first group, and thus that the intruder is able to share a (different) key simultaneously with all members of the second group (from which he is normally excluded).

To conclude, the implicit key authentication property seems to become very problematic in this protocol as soon as the intruder is a member of a group and is intended to be excluded from another non disjoint group.

We will now analyze the other security properties, considering that the intruder is not a member of any group.

5.2. Perfect Forward Secrecy

We will divide our analysis into two parts, considering successively the two flavors of perfect forward secrecy that we described in Section 3.

5.2.1. Complete Forward Secrecy

In the study of this property, we will assume that a session of the protocol has been executed in the past, and that E_I contains all long-term keys K_{in}. To simplify our writings, we consider anew a system with one session of four honest participants, but the results can be easily extended to a group of size n.

Since the session was executed in the past without any manipulation, the intruder only receives a set of values corresponding to all transmitted elements of $\mathcal{G}$. Furthermore, the group key computed by the different users is not the result of the exponentiation of an unknown element with a secret value: since the intruder cannot influence this session, each user is computing the same key: $\alpha^{r_1r_2r_3r_4}$. The sets of interest are therefore:

$$\begin{aligned}
\mathsf{S} &= \{r_1, r_2, r_1r_2, r_2r_3, r_1r_3, r_1r_2r_3, \\
&\quad\ r_2r_3r_4K_{14}, r_1r_3r_4K_{24}, r_1r_2r_4K_{34}\} \\
\mathsf{E}_I &= \{K_{14}, K_{24}, K_{34}\} \\
\mathsf{P}_S &= \{r_1r_2r_3r_4\}
\end{aligned}$$

The second step of our proof scheme will allow us to delete all long-term keys from these sets. There are many solutions to the corresponding linear system (e.g. $r_2 = 1$, $r_1r_3r_4 = 1$ and all other variables set to 0). We still have to check whether the conditions of Section 4.4.4 are verified for one of them. However, Condition 2 on the use of the services imposes that the only elements of P_I that can be obtained from S are those corresponding to elements of S or to one element of S multiplied

by the inverse of another one (since all services are starting services) and it can be checked that $r_1r_2r_3r_4$ is not equal to any element of S nor to the "substraction" (i.e. the composition in opposite directions) of two of them. It is therefore impossible to build an attack for this system.

It can be easily verified that considering several sessions with any number of participants does not help to obtain a solution to this problem. So, if the intruder is not allowed to participate to any session, the complete forward secrecy property is verified from our model's point of view.

5.2.2. Individual Forward Secrecy

In the previous section, we considered that the sessions executed before the compromise of the long-term keys had not been manipulated by the intruder. Now, we will consider that the intruder is able to manipulate the previous sessions for one (or a few) group participants leaving these sessions correct for the other ones.

For simplicity purposes, we will anew assume a system containing only one session with four participants: M_1, M_2, M_3 and M_4. We will assume that the services can be exploited and that the long-term keys are compromised. So, the sets of interest are:

$$\begin{aligned} \mathsf{S} &= \{r_1, r_2, r_3, r_4K_{14}, r_4K_{24}, r_4K_{34}\} \\ \mathsf{E}_I &= \{K_{14}, K_{24}, K_{34}\} \\ \mathsf{P}_S &= \{r_1K_{14}^{-1}, r_2K_{24}^{-1}, r_3K_{34}^{-1}, r_4\} \end{aligned}$$

If we delete the keys from these sets (as described in the second step of our proof-scheme), we can find that for each secret r_i, the resulting linear system has a trivial solution: $r_i = 1$. These solutions meet all conditions described above, and we can then assume that the individual forward secrecy is somehow suspicious.

We first consider the solution $r_4 = 1$ that corresponds to the secret of M_4. This service is in fact offered three times: when M_4 forms the broadcast by exponentiating the terms he receives with r_4K_{14}, r_4K_{24} and r_4K_{34}. These three services can be used indifferently, and we will choose to use the first one. So, the intruder replaces the term that M_4 will use to compute its view of the key with any element of $\mathcal{G}$ (say α^x) and replaces the term that M_4 will exponentiate with r_4K_{14} by the same element. A representation of this scenario where the intruder has chosen α^x to be $\alpha^{r_1r_2r_3}$ can be found in Fig. 1.7.

As we required, this manipulation only affects the key computed by M_1 (that will be $\alpha^{r_1^2r_2r_3r_4}$) while the other group members are still computing the same key: $\alpha^{r_1r_2r_3r_4}$. Finally, if K_{14} is compromised, the intruder is able to compute this key from the term $\alpha^{r_1r_2r_3r_4K_{14}}$ sent by M_4. The individual forward secrecy property is therefore not verified.

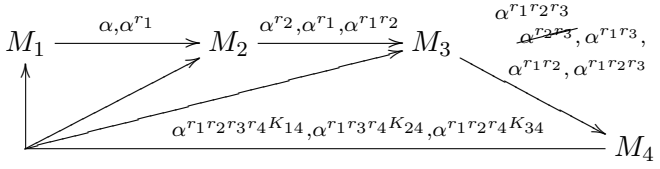


FIGURE 1.7. Attack against Individual Forward Secrecy

The fact that this attack provides the key computed by group-members other than M_4 corresponds to the exploitation of solutions of the type $r_i = 1$, $r_4(K_{i4}) = -1$, $r_4(K_{14}) = 1$ that are less trivial solutions of the system corresponding to the secret of M_i.

We can now check the solutions of the linear equations corresponding to the secrets of M_1, M_2 and M_3. Besides the solution discovered when exploiting the solution $r_4 = 1$, we can also use the solutions $r_i = 1 (1 \leq i < 4)$ (and all other services unused). So, if the intruder wants to attack M_i, he simply has to replace the term that M_i will exponentiate with $r_i K_{i4}^{-1}$ by any term that M_i previously exponentiated with r_i. An example of this scenario where $i = 2$ is represented in Fig. 1.8.

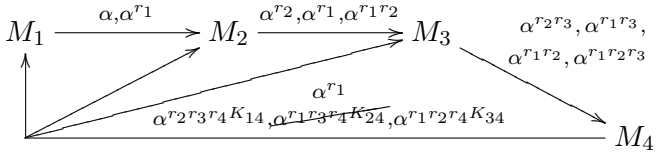


FIGURE 1.8. Attack against Individual Forward Secrecy

It can however be observed that this time M_i is computing a key that he does not share with anyone. So, even if this scenario can be performed in parallel against several group members, it appears to be less awkward than the previous one.

5.3. Resistance to Known-Key Attacks

This property expresses that the compromising of session keys does not allow a passive adversary to compromise keys of other sessions nor an active adversary to impersonate one of the protocol parties. The part of this property concerning the passive adversary is studied in [7] and we will focus on the second part. However the authors of [7] claim that the resistance to an active adversary is more dubious and suggest an attack that does not seem very useful in practice.

We will apply our method to the verification of this property as follows: we are assuming two sessions of the protocol with the same four participants. The random numbers generated by M_i during the first and second sessions of the protocol will be denoted r_i and r'_i respectively. In order to model the compromise of the session keys (rather than the compromise of session secrets), we will assume that the users are providing services corresponding to the secrets of the first session: $r_i K_{i4}^{-1}$ and r_4. Hence we can write that

$$\begin{aligned}
\mathsf{S} &= \{r_1, r_2, r_3, r_4K_{14}, r_4K_{24}, r_4K_{34}, \\
&\quad r'_1, r'_2, r'_3, r'_4K_{14}, r'_4K_{24}, r'_4K_{34}, \\
&\quad r_1K_{14}^{-1}, r_2K_{24}^{-1}, r_3K_{34}^{-1}, r_4\} \\
\mathsf{E}_I &= \emptyset \\
\mathsf{P}_S &= \{r'_1K_{14}^{-1}, r'_2K_{24}^{-1}, r'_3K_{34}^{-1}, r'_4\}
\end{aligned}$$

The linear system corresponding to $r'_iK_{in}^{-1} (1 \leq i < 4)$ is

$$\begin{aligned}
r_j + r_jK_{j4}^{-1} = 0 \qquad r'_j &= \delta(i,j) \\
r_4K_{14} + r_4K_{24} + r_4K_{34} + r_4 &= 0 \\
r'_4K_{14} + r'_4K_{24} + r'_4K_{34} &= 0 \\
r_4K_{j4} + r'_4K_{j4} - r_jK_{j4}^{-1} &= -\delta(i,j)
\end{aligned}$$

where $1 \leq j < 4$ and $\delta(i,j) = 1$ if $i = j$ and 0 otherwise. It can be checked that:

$$r_i = -1, r_iK_{i4}^{-1} = 1, r'_i = 1$$

and all other services being unused, is a solution.

If $i = 1$, it is however impossible to find an attack scheme: this solution would need the service $r_1K_{14}^{-1}$ (i.e. the service corresponding to the key computed by M_1 during the first session) to be applied on the value $\alpha^{r'_1}$ that is provided as a starting service during the second session. Nevertheless, for all other values of i, the following attack is possible:

(1) Let α^x be one of the input terms of the i-th round of the first protocol run. M_i will therefore send α^{xr_i}.
(2) The intruder replaces then the term $\alpha^{\frac{r_1 \ldots r_4}{r_i} K_{in}}$ with α^x. The group key computed by M_i will therefore be equal to $\alpha^{xr_iK_{in}^{-1}}$ and, since we study known-key attacks, we assume that this value will be compromised.
(3) In the second run of the protocol, the intruder replaces one of the i-th round inputs with $\alpha^{xr_iK_{in}^{-1}}$. M_i will therefore send $\alpha^{xr_iK_{in}^{-1}r'_i}$.

(4) In the broadcast of the second run of the protocol, the intruder finally replaces the term intended to M_i with α^{xr_i} (obtained in the first step of our scenario). Hence $S'_n(M_i)$ will be computed as $\alpha^{xr_iK_{in}^{-1}r'_i}$ that has been obtained during the third step of our scenario.

At the end of this scenario, the intruder will possess a key that M_i believes to be secret. However this key is unknown to the rest of the group and the compromised key used is a malformed key which reduces the scope of these attacks. However, if all malformed keys are available, the intruder can perform this attack simultaneously against almost all the members of the group.

We can now turn to the secrecy of r'_n. If we look at the linear system corresponding to this secret (i.e. the system we just described where the right part of each equation has been updated), we can find two types of solutions. The first one is:

$$r_i = -1, r_iK_{in}^{-1} = 1, r'_nK_{in} = 1$$

From these solutions, we can obtain the scenarios corresponding to the attack proposed in [7]. The scope of these attacks is the same as the one we just described.

However another type of solution of the linear system can be found:

$$r_n = 1, r_nK_{in} = -1, r'_nK_{in} = 1$$

For $1 \leq i < 4$, it is possible to apply the following scenario:

(1) In the last round inputs of the protocol's first session, the intruder replaces $\alpha^{r_1 \cdots r_3/r_i}$ with $\alpha^{r_1r_2r_3}$. Hence all elements of the broadcast will be preserved except the one intended to M_i that will be equal to $\alpha^{r_1r_2r_3r_4K_{i4}}$. $S_4(M_4)$ will therefore be equal to $\alpha^{r_1r_2r_3r_4}$ and shared by all members of the group except M_i. In a context of known-key attacks, we will assume that this key is compromised.
(2) In the last round's inputs of the protocol's second session, the intruder will substitute $\alpha^{r'_1r'_2r'_3/r'_i}$ with $\alpha^{r_1r_2r_3r_4}$ and $\alpha^{r'_1r'_2r'_3}$ with $\alpha^{r_1r_2r_3r_4K_{i4}}$. Hence M_4 will broadcast $\alpha^{r_1r_2r_3r_4K_{i4}r'_4}$ and compute his view of the key as $S_4(M_4) = \alpha^{r_1r_2r_3r_4K_{i4}r'_4}$.

A representation of this attack for $i = 1$ can be found in Fig. 1.9. This scenario is more dangerous since we assume that a key, shared (and used normally) by all members of the group except one is compromised. Finally, the resistance of this protocol to known-key attacks appears to be more problematic than expected in [7].

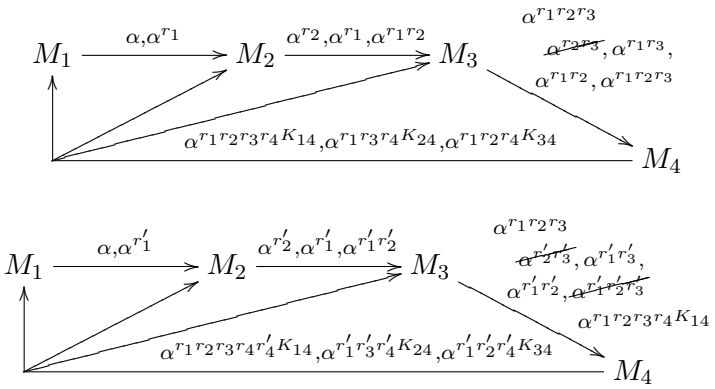


FIGURE 1.9. Attack against Resistance to Known Keys

6. Considering the A-GDH.2-MA Protocol

Very often, AGKAP's are defined together with a number of "sub-protocols" enabling dynamic changes in group constitution [7], [14], [41]. It is the case of the A-GDH.2 protocol that is proposed to be used in parallel with other protocols enabling the addition of new members in the group (A-GDH.2-MA), the removal of a member, the fusion of two groups, etc.

In this section, we consider the use of the A-GDH.2-MA protocol, the other protocols of the suite not being precisely defined in the literature.

6.1. Definition of the A-GDH.2-MA Protocol

This protocol is defined as follows:

Protocol 3 : A-GDH.2-MA Protocol

The A-GDH.2-MA protocol allows a group M of n users $M_1, \ldots, M_n$ sharing a key generated through the A-GDH.2 protocol to add a new member to the group, M_{n+1}, and to share a key with him. We assume α and $\mathcal{G}$ to be defined as in the context of the A-GDH.2 protocol.

The A-GDH.2-MA protocol executes in 2 rounds: in the first one, M_n sends to M_{n+1} a message computed from the one he broadcast in the last round of the A-GDH.2 protocol and from the old key while in the second round, M_{n+1} broadcasts the new keying material to the group. The actual protocol is as follows:

Round 1:

1. M_n selects $\hat{r}_n \in \mathbb{Z}_q^*$
2. $M_n \rightarrow M_{n+1} : \{\alpha^{\hat{r}_n \frac{r_1 \ldots r_n}{r_i}} K_{in} | i \in [1,n]\}, \alpha^{\hat{r}_n r_1 \ldots r_n}$

Round 2:
1. M_{n+1} selects $r_{n+1} \in \mathbb{Z}_q^*$
2. $M_{n+1} \rightarrow$ All M_i: $\{\alpha^{\hat{r}_n \frac{r_1 \cdots r_{n+1}}{r_i}.K_{in}.K_{in+1}} | i \in [1, n+1]\}$

Upon receipt of the above, every M_i computes the new group key as:

$$\begin{aligned} S_{n+1} &= \alpha^{(\hat{r}_n \frac{r_1 \cdots r_{n+1}}{r_i}.K_{in}.K_{in+1}).K_{in}^{-1}.K_{in+1}^{-1}.r_i} \\ &= \alpha^{\hat{r}_n r_1 \ldots r_{n+1}} \end{aligned}$$

The security properties of the A-GDH.2 protocol have to be preserved after execution of this protocol. Fig. 1.10 represents a protocol run where a fifth member is added to the group represented in Fig. 1.5.

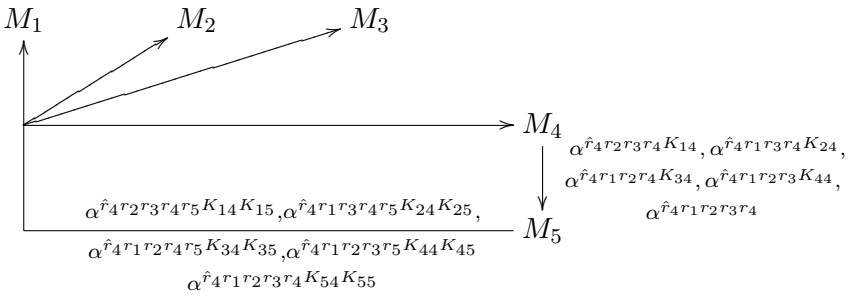


FIGURE 1.10. A-GDH.2-MA Protocol Run

We can observe on this figure that the first three elements of the first round of this protocol are those of the broadcast of the setup protocol exponentiated with $\hat{r}_4$.

6.2. Analysis of the A-GDH.2-MA Protocol

In this paragraph, we will extend our analysis of the A-GDH.2 protocol taking into account the presence of the Member Adding protocol. As a first step, we will study the Implicit Key Authentication property and consider two sessions of the protocols: in the first session, the A-GDH.2 protocol is executed by M_1, M_2 and M_3; while in the second session a fourth member, M_4, is added to this group. Following the same approach as above, we will first write the sets S, E_I and P_S that

will be the union of those corresponding to the sessions of each protocol:

$$\begin{aligned}
\mathsf{S} &= \{r_1, r_2, r_3K_{13}, r_3K_{23},\\
&\quad r_3\hat{r}_3, r_3\hat{r}_3K_{13}, r_3\hat{r}_3K_{23}, r_3\hat{r}_3K_{33},\\
&\quad r_4K_{14}, r_4K_{24}, r_4K_{34}, K_{43}K_{44}\}\\
\mathsf{E}_I &= \emptyset\\
\mathsf{P}_S &= \{r_1K_{13}^{-1}, r_2K_{23}^{-1}, r_3, r_1K_{13}^{-1}K_{14}^{-1}, r_2K_{23}^{-1}K_{24}^{-1},\\
&\quad r_3K_{33}^{-1}K_{34}^{-1}, r_4K_{43}^{-1}K_{44}^{-1}\}
\end{aligned}$$

The first line of the expression of S corresponds to the execution of the key-generation protocol, the second to the first round of the A-GDH.2-MA protocol, and the last to the second round of the A-GDH.2-MA protocol. The first three elements of P_S correspond to the secrets of the key generation protocol while the others correspond to secrets of the member adding protocol.

E_I being empty, we can immediately study the linear system corresponding to the secrets. This system is a little longer than the previous one but remains quite regular. We give its expression for the verification of the secrecy of $r_1K_{13}^{-1}$.

$$\begin{aligned}
r_1 = 1 \qquad r_2 &= 0\\
r_3K_{13} + r_3K_{23} + r_3\hat{r}_3 + r_3\hat{r}_3K_{13} + r_3\hat{r}_3K_{23} + r_3\hat{r}_3K_{33} &= 0\\
r_4K_{14} + r_4K_{24} + r_4K_{34} &= 0\\
r_3\hat{r}_3 + r_3\hat{r}_3K_{13} + r_3\hat{r}_3K_{23}r_3\hat{r}_3K_{33} &= 0\\
r_3K_{13} + r_3\hat{r}_3K_{13} = -1 \qquad r_3K_{23} + r_3\hat{r}_3K_{23} &= 0\\
r_3\hat{r}_3K_{33} &= 0\\
r_4K_{14} = -1 \qquad r_4K_{24} = 0 \qquad r_4K_{34} = 0 \qquad K_{43}K_{44} &= 0
\end{aligned}$$

If we solve this system, we find that $r_1K_{13}^{-1}$ and $r_2K_{23}^{-1}$ can be compromised: they can be obtained by combining the services r_i, $r_3\hat{r}_3$ and $r_3\hat{r}_3K_{i3}$ (for $i = 1, 2$). The other secrets cannot be compromised in this scheme: the solutions corresponding to r_3 do not respect the conditions on the use of the services; and the system has no solution for the other secrets. The scenario corresponding to $r_iK_{i3}^{-1}$ is as follows:

(1) M_1, M_2 and M_3 execute the key-generation protocol, but the intruder intercepts the broadcast of the last round.
(2) The intruder obtains that M_n starts the A-GDH.2-MA protocol with another user of the network, and intercepts the first message.

(3) The intruder sends the parts of this message corresponding to the users M_1 and M_2, faking the broadcast of the A-GDH.2 protocol.

When it is done, M_1 and M_2 are sharing with the intruder the key $\alpha^{r_1 r_2 r_3 \hat{r}_3}$ that has been sent by M_3 as the last part of the first message of the A-GDH.2-MA protocol. A scheme corresponding to this attack is represented in Fig. 1.11.

To conclude, the Implicit Key Authentication property seems to become even more problematic when we consider the possibility of the use of the A-GDH.2-MA protocol in parallel with the A-GDH.2 protocol: the intruder does not have to be intended to take part to any session if he wants to defeat the IKA property. The other security properties could be similarly analyzed.

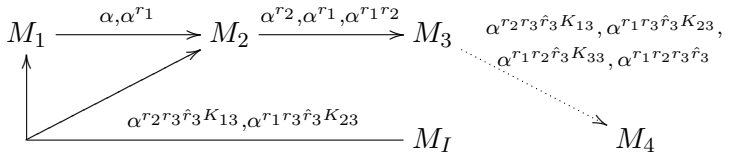


FIGURE 1.11. Attack considering the A-GDH.2-MA Protocol

7. Analysis of the SA-GDH.2 Protocol

In the previous sections, we used our machinery to analyze the A-GDH.2 protocol suite and obtained attacks against all claimed security properties. These attack schemes are very systematic and can be exploited in a large range of protocols that have been (or could be) defined using the A-GDH scheme.

We will illustrate this intuition with an attack against the SA-GDH.2 protocol [7]. This protocol is an extension of the A-GDH.2 one providing *complete group authentication*, i.e. guaranteeing that two group members M_i and M_j compute the same key $S_{i,j}$ only if $S_{i,j}$ has been contributed to by every $M_p \in \mathsf{M}$ (assuming that M_i and M_j have the same view of the group membership). The main change in the requirements for this protocol is in the definition of the long-term keys: it is assumed that each user M_i shares two keys (one for each direction of communication) with every other user M_j: for every ordered pair $< i, j >$ $(1 \leq i \neq j \leq n)$, we use $< K_{ij}, K_{ij}^{-1} >$ to denote the unidirectional key shared by M_i and M_j and its inverse, respectively.

Protocol 4 : SA-GDH.2 protocol

This protocol allows a group M of n users $M_1, \ldots, M_n$ arranged into a ring to share a key. We assume α and $\mathcal{G}$ to be defined as in the context

of the A-GDH.2 protocol and K_{ij} to be a long-term unidirectional secret key shared by M_i and M_j.

As for A-GDH.2, this protocol executes in n rounds; the $n-1$ first ones collecting the key contributions, while the last one is used to broadcast the keying material. It is as follows:

Round i $(1 \leq i \leq n)$:
1. M_i selects $r_i \in \mathbb{Z}_q^*$
2. M_i receives a set of n intermediate values: $\{V_k | 1 \leq k \leq n\}$ (M_1 can be thought of as receiving an empty set in the first round and, in the initial round, M_1 sets $V_1 = \alpha$):

$$V_k = \begin{cases} \alpha^{\frac{r_1 \ldots r_{i-1}}{r_k} \cdot (K_{1k} \ldots K_{(i-1)k})} & \text{if } k \leq (i-1) \\ \alpha^{r_1 \ldots r_{i-1} \cdot (K_{1k} \ldots K_{(i-1)k})} & \text{if } k > (i-1) \end{cases}$$

3. M_i updates each V_k as follows:

$$V_k = \begin{cases} (V_k)^{K_{ik} r_i} = \alpha^{\frac{r_1 \ldots r_i}{r_k} \cdot (K_{1k} \ldots K_{ik})} & \text{if } k < i \\ (V_k)^{K_{ik} r_i} = \alpha^{r_1 \ldots r_i \cdot (K_{1k} \ldots K_{ik})} & \text{if } k > i \\ V_k & \text{if } k = i \end{cases}$$

4. M_i sends the updated V_k to M_{i+1} if $i < n$ or broadcast $V_1 \ldots V_{n-1}$ if $i = n$.

Upon receipt of the above, every M_i selects the appropriate V_i where :

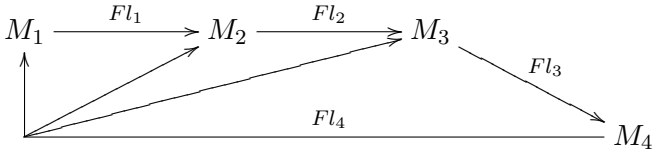
$$V_i = \alpha^{\frac{r_1 \ldots r_n}{r_i} \cdot (K_{1i} \ldots K_{ni})}$$

and computes:

$$S_n = V_i^{(K_{1i}^{-1} \ldots K_{ni}^{-1}) \cdot r_i} = \alpha^{r_1 \ldots r_n}$$

In order to better understand the structure of this protocol, we represented a protocol run with four participants in Fig. 1.12.

We can now examine whether the implicit key authentication property is verified by using our method the same way we used it with the A-GDH.2 protocol: we will assume a first session with group constitution M_1, M_I, M_2, M_3 and a second one with group constitution M_1, M_2, M_3. If we use the letters r_i and r_i' to denote the random contributions generated by the user M_i during the first and the second sessions respectively,



$$
\begin{aligned}
Fl_1 &= \alpha, \alpha^{r_1 K_{12}}, \alpha^{r_1 K_{13}}, \alpha^{r_1 K_{14}} \\
Fl_2 &= \alpha^{r_2 K_{21}}, \alpha^{r_1 K_{12}}, \alpha^{r_1 r_2 K_{13} K_{23}}, \alpha^{r_1 r_2 K_{14} K_{24}} \\
Fl_3 &= \alpha^{r_2 r_3 K_{21} K_{31}}, \alpha^{r_1 r_3 K_{12} K_{32}}, \alpha^{r_1 r_2 K_{13} K_{23}}, \alpha^{r_1 r_2 r_3 K_{14} K_{24} K_{34}} \\
Fl_4 &= \alpha^{r_2 r_3 r_4 K_{21} K_{31} K_{41}}, \alpha^{r_1 r_3 r_4 K_{12} K_{32} K_{42}}, \alpha^{r_1 r_2 r_4 K_{13} K_{23} K_{43}}
\end{aligned}
$$

FIGURE 1.12. SA-GDH.2 Protocol Run with 4 Participants

we can describe this configuration as:

$$
\begin{aligned}
\mathsf{S} &= \{r_1 K_{1I}, r_1 K_{12}, r_1 K_{13}, r_I K_{I1}, r_I K_{I2}, r_I K_{I3}, \\
&\quad r_2 K_{21}, r_2 K_{2I}, r_2 K_{23}, r_3 K_{31}, r_3 K_{3I}, r_3 K_{32}, \\
&\quad r'_1 K_{12}, r'_1 K_{13}, r'_2 K_{21}, r'_2 K_{23}, r'_3 K_{31}, r'_3 K_{32}\} \\
\mathsf{E}_I &= \{r_I, K_{1I}, K_{2I}, K_{3I}, K_{I1}, , K_{I2}, K_{I3}\} \\
\mathsf{P}_S &= \{r'_1 K_{21}^{-1} K_{31}^{-1}, r'_2 K_{12}^{-1} K_{32}^{-1}, r'_3 K_{13}^{-1} K_{23}^{-1}\}
\end{aligned}
$$

If we look at the equation system for the secret of M_2 for instance, we can obtain that a solution allowing to obtain the secret $r'_2 K_{12}^{-1} K_{32}^{-1}$ is:

$$r'_2 K_{21} = r_2(K_{2I}) = r_1(K_{1I}) = r_3(K_{3I}) = 1$$
$$r_2 K_{21} = r_1 K_{12} = r_3 K_{32} = -1$$

This solution meets all constraints defined in Section 4.4.4, and can be used to obtain the attack scenario described in Fig. 1.13.

This figure shows the values exchanged by the different group members ordered by the indexes of V. During his round of the first session, the intruder (who is the second member of the group) replaces V_3 with $\alpha^{r_1 K_{12}}$ and V_2 with α^{r_1}, so M_2 provides $\alpha^{r_1 r_2 K_{2I}}$ and $\alpha^{r_1 r_2 K_{12} K_{23}}$. M_I will put this last value in place of V_2 in the input message of M_3, so M_3 will send $\alpha^{r_1 r_2 r_3 K_{2I} K_{3I}}$ and $\alpha^{r_1 r_2 r_3 K_{12} K_{23} K_{32}}$. These last two values will be used in the second session: in the first message of this session, M_I will replace V_2 (i.e. the element intended to M_3) with $\alpha^{r_1 r_2 r_3}$ that he computed from $\alpha^{r_1 r_2 r_3 K_{2I} K_{3I}}$. M_2 will therefore update (and send) V_2 as $\alpha^{r_1 r_2 r_3 r'_2 K_{23}}$. Finally, in the last broadcast, M_I will replace V_2 with $\alpha^{r_1 r_2 r_3 K_{12} K_{23} K_{32}}$ and M_2 will compute his view of the group key as $\alpha^{r_1 r_2 r_3 r'_2 K_{23}}$, value that he sent during the previous round.

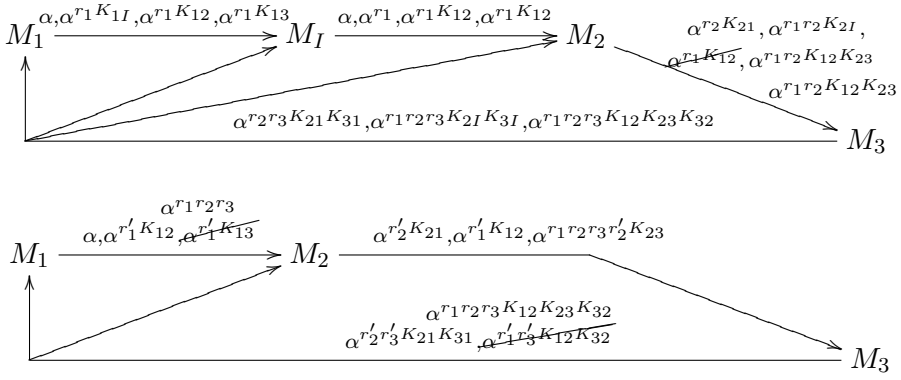


FIGURE 1.13. Attack against the SA-GDH.2 Protocol

We can see that this key was obtained in the same way as in the case of the A-GDH.2 protocol: the random contribution to the key was obtained from the message M_2 emits during the second session, while the ratios corresponding to the long-term keys included in M_2's secret were obtained during the session of which the intruder is a regular member.

8. Concluding Remarks

In this chapter, we presented a number of attacks against authenticated group key agreement protocols. Most of them are coming from the fact that the properties of group protocols are not trivial extensions of the ones of two-parties protocols: for instance, the fact that a group member computes a bad key can remain unnoticed by the other group members, particularly if the group is large, while it prevents any exchange of messages when only two users are considered. The discovery of all these vulnerabilities emphasizes the need of using systematic methods for reasoning about security protocols.

Due to their particular structure, the analysis of the GDH protocols (A-GDH.2 and SA-GDH.2) required the development of a new machinery. We discovered that within some reasonable transposition of the idealizations usually stated in the context of logical approaches for protocol analysis, and by adequately modelling the operations performed by the honest users during the protocol sessions, it was particularly easy to verify the security properties of these protocols since it came down to solving a linear equation system.

Trying to draw lessons from the weaknesses we encountered, we tried to develop some fixes for these protocols. This question is studied in the next chapter.

CHAPTER 2

A Fix for the A-GDH Protocols?

1. Introduction

In the previous chapter, we presented several flaws in the A-GDH.2 and SA-GDH.2 protocols that are part of the Cliques protocols suites. A natural question that came at the end of this chapter was: "How could we fix these protocols?".

In order to answer this question, we tried to modify the A-GDH protocols by modifying the content of the messages, then by adopting different orders to exchange them, but we were always able to find attacks against our candidates. Finally, we wondered whether it was possible to write a protocol based on the same design assumptions than the A-GDH ones and providing implicit key authentication. The answer appears to be "no" and this is what we will explain in this chapter.

The next sections are organized as follows. In Section 2, we define a class of protocols generalizing those analyzed in the previous chapter: the GDH-Protocols. The protocols of this family are those we consider to be possible fixes for the A-GDH protocols: we think that protocols outside this family would be too different from the initial ones to be presented as fixes. Together with this definition, we are introducing a number of concepts that are transpositions of ideas presented in the model of the previous chapter.

All these concepts will be used in Section 3 to write a number of properties GDH-Protocols must present as a consequence of their definition.

In Section 4, we use these properties to show that it is always possible to write the computation a group member executes to obtain the group key as a product of services provided by the different protocol participants.

Finally, in Section 5, we show that it is always possible for an active attacker to combine these services in order to fool at least one member of a group from which he is excluded into accepting a compromised value as group key, provided that the size of the group executing the protocol is at least four.

2. A Family of Protocols

We are considering a particular class of Authenticated Group Key Agreement Protocols. The goal of these protocols is the sharing of a session key $\alpha^{r_1 \cdots r_n}$ that will be computed by exponentiating an element of a group $\mathcal{G}$ with a product of random numbers and keys. The implicit key authentication property states that this session key has to remain out of reach of any user $M_I \notin \mathsf{M}$ (where M is the set of the group members), considering the presence of an active attacker that can be a legitimate member of some groups.

The protocols of the family we consider are executed in a very regular way: the behavior of the principals can be described as a sequence of exponentiation of elements $\alpha^x \in \mathcal{G}$ with random numbers and keys.

As in the previous chapter, we use the letter r_i to denote M_i's contribution to the group key, and the letter K_{ij} to denote a long-term key shared by M_i and M_j. Since these values are never sent on the network on a readable form, we assume that the random contributions are only known by the user who generated them, and that the long-term keys are only known by the two users who are intended to share them. Furthermore, we assume that the different values (random numbers and keys) used for exponentiation in these protocols do not present any arithmetic relation between them.

We also assume that these protocols are constant under member substitution: substituting member M_i with a user M_j in the group constitution will only change the protocol execution by substituting M_i with M_j and keys of the form K_{ik} with K_{jk}. This assumption excludes protocols the definition of which would contain rules such as: "User M_i exponentiates the term intended to M_j with K_{ij}^x where x is the last bit of M_j's identifier" for instance.

As an example of protocol of the family we consider, we suggest a protocol written for the purpose of illustrating the definitions, propositions and theorems we will develop in the next sections.

Example 2.1 We describe here a protocol in a similar form as the one commonly used in the literature and in [7] for instance. This protocol allows a group of users M_1, M_2 and M_3 to contributively generate a key $\alpha^{r_1 r_2 r_3}$. Through the rest of this chapter, we will call this protocol the Ex-GDH protocol.

Let $\mathcal{G}$ be a cyclic subgroup of prime order q, and α be a generator of $\mathcal{G}$. Let $r_i, \hat{r}_i \in \mathbb{Z}_p^*$ be random values generated by M_i and K_{ij} be a secret long term key shared by M_i and M_j. The three group members M_1, M_2

and M_3 generate the group key by exchanging the following messages:

$$\begin{aligned} M_1 \to M_2 &: \alpha^{\hat{r}_1}, \alpha^{r_1} \\ M_2 \to M_3 &: \alpha^{\hat{r}_1 r_2 K_{23}}, \alpha^{r_1 K_{23}}, \alpha^{r_1 r_2} \\ M_3 \to M_1, M_2 &: \alpha^{\hat{r}_1 r_2 r_3 K_{13}}, \alpha^{r_1 r_3 K_{23}^2} \end{aligned}$$

Upon receipt of the above, M_1 computes the group key $\alpha^{r_1 r_2 r_3}$ from $\alpha^{\hat{r}_1 r_2 r_3 K_{13}}$, M_2 from $\alpha^{r_1 r_3 K_{23}^2}$ and M_3 from $\alpha^{r_1 r_2}$.

We now start the presentation of our modelling of these protocols with some definitions.

Definition 2.1 *Let:*

(1) M *be a set of* n *group members* $\{M_1, \ldots, M_n\}$ *from which the intruder is excluded;*
(2) R *be the set the elements of which represent random values generated during the protocol execution,* $\mathsf{R}_i \subset \mathsf{R}$ *denoting the set of random values generated by* M_i*;*
(3) K *be the set the elements of which represent the long-term shared keys,* $\mathsf{K}_i \subset \mathsf{K}$ *denoting the set of keys known by* M_i *and* $K_{ij} \in (\mathsf{K}_i \cap \mathsf{K}_j)$ *a key shared by* M_i *and* M_j *(for the simplicity of the notations, we will assume that* $K_{ij} = K_{ji}$ *and occasionally write* K_{M_i} *instead of* K_i *or* $K_{M_i M_j}$ *instead of* K_{ij}*);*
(4) atoms *be elements of* $\mathsf{R} \cup \mathsf{K}$
(5) $(\overline{\mathsf{R}}, \cdot)$ *and* $(\overline{\mathsf{K}}, \cdot)$ *be the commutative groups freely generated from* R *and* K *respectively. The unit element of these groups is denoted* 1*. For simplicity, we use multiplicative notations and often write* $a \cdot b$ *as* ab *and* $a \cdot a$ *as* a^2*;*
(6) $(\mathsf{P}, \cdot)$ *be the commutative group isomorphic to* $(\overline{\mathsf{R}} \times \overline{\mathsf{K}})$ *through the morphism* $f(r, K) = r \cdot K$*. It can be noticed from this definition that* P *is free;*
(7) p_{R} *and* p_{K} *be the two elements of* P *such that* $p = p_{\mathsf{R}} \cdot p_{\mathsf{K}}$*, with* $p_{\mathsf{R}} \in \bar{\mathsf{R}}$ *and* $p_{\mathsf{K}} \in \bar{\mathsf{K}}$*. Similarly,* p_a *denotes* a^e *where* $p = a^e a_1^{e_1} \cdots a_n^{e_n}$*,* $a \neq (a_i)_{i=1\ldots n}$*, and* $a, (a_i)_{i=1\ldots n}$ *are atoms;*
(8) G *be the set that models the finite group* $\mathcal{G}$*. This set is defined through a bijection* $\mathbf{alphaexp} : \mathsf{P} \to \mathsf{G}$ *that represents the exponentiation of the public group generator* α *with some product of random values and keys;*
(9) $\alpha = \mathbf{alphaexp}(1)$ *be the symbolic representation of the publicly known generator of* $\mathcal{G}$*.* $\mathbf{alphaexp}(p)$ *will typically be denoted* α^p*;*
(10) $\mathbf{exp} : \mathsf{G} \times \mathsf{P} \to \mathsf{G}$ *that represents the exponentiation of an element of* G *with an element of* P*. If* $g \in \mathsf{G}$ *and* $p \in \mathsf{P}$*,* $\mathbf{exp}(g, p) = \mathbf{alphaexp}(\mathbf{alphaexp}^{-1}(g) \cdot p)$*.*

We illustrate these definitions through the following example.

Example 2.2 In our Ex-GDH protocol, $\mathsf{M} = \{M_1, M_2, M_3\}$, $\{r_1, \hat{r}_1, r_2, r_3\} \subset \mathsf{R}$ and $\{K_{13}, K_{23}\} \subset \mathsf{K}$; $p = r_1 \cdot r_3 \cdot K_{23}^2$ is an element of P, $p_{\mathsf{R}} = r_1 \cdot r_3$, $p_{\mathsf{K}} = K_{23}^2$, $p_{K_{23}} = K_{23}^2$, $p_{r_2} = 1$ and $\mathbf{exp}(\alpha^{r_1 \cdot r_3 \cdot K_{23}}, K_{23}) = \alpha^{r_1 \cdot r_3 \cdot K_{23}^2}$.

As it can be seen, we do not take any arithmetic relation that could exist between elements of R and K into account. It can also be observed that according to our definitions, the set G is infinite (while $\mathcal{G}$ is a finite group of prime order).

The messages of the protocols we consider are all constituted of sequences of elements of $\mathcal{G}$ (modelled as elements of G). We call such sequences *GDH-Terms* and denote the set of GDH-Terms as A.

Definition 2.2 *Let A be the set of finite sequences $\langle g_1, \ldots, g_n \rangle$ of elements of G. An element of A will be called a* GDH-Term. *The i-th element of $t \in \mathsf{A}$ will be noted $t(i) \in \mathsf{G}$.*

We define a subterm relation $\sqsubset$ as follows :

Definition 2.3 *Let a be an atom.*

- *$a \sqsubset p \in \mathsf{P}$ if p can be written as $a^e a_1^{e_1} \ldots a_n^{e_n}$ where $(a_i)_{i=1\ldots n}$ are atoms, $(a_i)_{i=1\ldots n} \neq a$ and $e \neq 0$;*
- *$a \sqsubset g \in \mathsf{G}$ iff $a \sqsubset \mathbf{alphaexp}^{-1}(g)$.*
- *$a \sqsubset t \in \mathsf{A}$ iff $\exists i : a \sqsubset t(i)$.*

If $a \sqsubset x$, we say that a is a subterm of x or that x contains a.

The following example illustrates these definitions.

Example 2.3 Let $t = \langle \alpha^{r_1 r_2}, \alpha^{r_1 K_{23}} \rangle$ be an element of A. $t(2) = \alpha^{r_1 K_{23}} \in \mathsf{G}$, $r_1 \sqsubset t$ and $K_{23} \not\sqsubset t(1)$.

In order to describe our protocols, we now exploit a part of the strand-space definitions ([32], Def. 1-5 and Prop. 1, pp. 234-236).

Definition 2.4 *A signed term is a pair $\langle \sigma, t \rangle$ with $t \in \mathsf{A}$ and σ is one of the symbols $+, -$. We will write a signed term as $+t$ or $-t$. $(\pm\mathsf{A})^*$ is the set of finite sequences of signed terms. We will denote a typical element of $(\pm\mathsf{A})^*$ by $\langle \langle \sigma_1, t_1 \rangle, \ldots, \langle \sigma_n, t_n \rangle \rangle$ or in a shorter way by $\langle \sigma_1 t_1, \ldots, \sigma_n t_n \rangle$.*

We show an instance of strand in the following example.

Example 2.4 $\langle \langle +, \alpha^{\hat{r}_1} \alpha^{r_1} \rangle, \langle -, \alpha^{\hat{r}_1 r_2 r_3 K_{13}} \alpha^{r_1 r_3 K_{23}^2} \rangle \rangle$ is a strand representing the role of M_1 in an execution of our Ex-GDH protocol. This notation will typically be abbreviated by $\langle +\alpha^{\hat{r}_1} \alpha^{r_1}, -\alpha^{\hat{r}_1 r_2 r_3 K_{13}} \alpha^{r_1 r_3 K_{23}} \rangle$.

A *strand space* over A is a set Σ with a trace mapping $\mathrm{tr} : \Sigma \to (\pm\mathsf{A})^*$. By abuse of language, we will still treat signed terms as ordinary terms. For instance, we shall refer to subterms of signed terms. We will usually represent a strand space by its underlying set of strands Σ.

Definition 2.5 *Fix a strand space Σ.*

(1) *A node is a pair $\langle s, i\rangle$, with $s \in \Sigma$ and i an integer satisfying $1 \leq i \leq length(tr(s))$. The set of nodes is denoted $\mathcal{N}$. We will say the node $\langle s, i\rangle$ belongs to strand s. Clearly, every node belongs to a unique strand.*

(2) *If $n = \langle s, i\rangle \in \mathcal{N}$ then $index(n) = i$ and $strand(n) = s$. Define $term(n)$ to be $(tr(s))(i)$, i.e. the i-th signed term in the trace of s. Similarly, $uns\_term(n)$ is $((tr(s))(i))_2$, i.e. the unsigned part of the i-th signed term in the trace of s.*

(3) *There is an edge $n_1 \to n_2$ if and only if $term(n_1) = +t$ and $term(n_2) = -t$ for some $t \in \mathsf{A}$. Intuitively, the edge means that n_1 sends the message t, which is received by n_2, recording a potential causal link between those strands.*

(4) *When $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i+1\rangle$ are members of $\mathcal{N}$, there is an edge $n_1 \Rightarrow n_2$. Intuitively, the edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. We write $n' \Rightarrow^+ n$ to mean that n' precedes n (not necessarily immediately) on the same strand.*

$\mathcal{N}$ together with both sets of edges $n_1 \to n_2$ and $n_1 \Rightarrow n_2$ is a directed graph $\langle \mathcal{N}, (\to \cup \Rightarrow)\rangle$.

The following example shows a strand space representing a session of our Ex-GDH protocol.

Example 2.5 Let s_1, s_2 and s_3 be three strands representing the roles of M_1, M_2 and M_3 in the Ex-GDH protocol. The corresponding strand space is represented in Fig. 2.1.

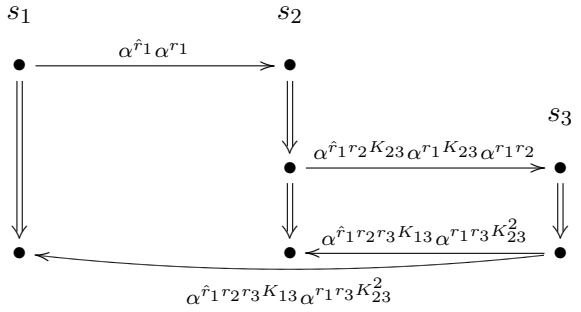


FIGURE 2.1. A run of the Ex-GDH protocol

It can be observed that s_1 corresponds to the strand mentioned in Example 2.4.

A *bundle* is a finite subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$ for which we can regard the edges as expressing the causal dependencies of the nodes.

Definition 2.6 *Suppose* $\rightarrow_{\mathcal{C}} \subset \rightarrow$*; suppose* $\Rightarrow_{\mathcal{C}} \subset \Rightarrow$*; and suppose* $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}}) \rangle$ *is a subgraph of* $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$*.* $\mathcal{C}$ *is a bundle if:*

(1) $\mathcal{N}_{\mathcal{C}}$ *and* $\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}}$ *are finite;*
(2) *if* $n_2 \in \mathcal{N}_{\mathcal{C}}$ *and* $term(n_2)$ *is negative, then there is a unique* n_1 *such that* $n_1 \rightarrow_{\mathcal{C}} n_2$*;*
(3) *if* $n_2 \in \mathcal{N}_{\mathcal{C}}$ *and* $n_1 \Rightarrow n_2$ *then* $n_1 \Rightarrow_{\mathcal{C}} n_2$*;*
(4) $\mathcal{C}$ *is acyclic.*

In conditions (2) and (3), it follows that $n_1 \in \mathcal{N}_{\mathcal{C}}$, because $\mathcal{C}$ is a graph.

Definition 2.7 *A node* n *is in a bundle* $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}}) \rangle$*, written* $n \in \mathcal{C}$*, if* $n \in \mathcal{N}_{\mathcal{C}}$*; a strand* s *is in* $\mathcal{C}$ *if all of its nodes are in* $\mathcal{N}_{\mathcal{C}}$*.*

If $\mathcal{C}$ *is a bundle, then the* $\mathcal{C}$*-height of a strand* s *is the largest* i *such that* $\langle s, i \rangle \in \mathcal{C}$*.*

Example 2.6 The scheme of Example 2.5 represents a bundle $\mathcal{C}$ and it remains a bundle if you suppress $\langle s_1, 2 \rangle$ from $\mathcal{N}_{\mathcal{C}}$ as well as the arrows leading to this node from $\rightarrow_{\mathcal{C}}$ and $\Rightarrow_{\mathcal{C}}$. However, it is not a bundle anymore if $\langle s_2, 1 \rangle$ and the arrows leading to and starting from this node are suppressed from $\mathcal{N}_{\mathcal{C}}$, $\rightarrow_{\mathcal{C}}$ and $\Rightarrow_{\mathcal{C}}$ since $\langle s_2, 2 \rangle \in \mathcal{C}$ and $\langle s_2, 1 \rangle \Rightarrow \langle s_2, 2 \rangle$.

Definition 2.8 *If* $\mathcal{S}$ *is a set of edges, i.e.* $\mathcal{S} \subset \rightarrow \cup \Rightarrow$*, then* $\prec_{\mathcal{S}}$ *is the transitive closure of* $\mathcal{S}$ *and* $\preceq_{\mathcal{S}}$ *is the reflexive, transitive closure of* $\mathcal{S}$*.*

The relations $\prec_{\mathcal{S}}$ and $\preceq_{\mathcal{S}}$ are each subsets of $\mathcal{N}_{\mathcal{S}} \times \mathcal{N}_{\mathcal{S}}$, where $\mathcal{N}_{\mathcal{S}}$ is the set of nodes incident with any edge in $\mathcal{S}$.

Proposition 2.1 *Suppose* $\mathcal{C}$ *is a bundle. Then* $\preceq_{\mathcal{C}}$ *is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in* $\mathcal{C}$ *has* $\preceq_{\mathcal{C}}$*-minimal members.*

We regard $\preceq_{\mathcal{C}}$ as expressing causal precedence, because $n \preceq_{\mathcal{C}} n'$ holds only when n's occurrence causally contributes to the occurrence of n'. When a bundle $\mathcal{C}$ is understood, we will simply write $\preceq$. Similarly, we will say that a node n *precedes* a node n' if $n \preceq n'$.

Considering a bundle allows us to understand the way messages are exchanged during a run of a protocol. However, it does not express how these messages are built, which is an important property for the class of protocols we are analysing. As explained in the literature concerning the

Cliques protocols, these protocols are executed in a very regular way: the group members receive sequences of elements of $\mathcal{G}$ and exponentiate these elements with products of known random values and keys to construct the messages they will send. So, for any element used by a group member to compute his view of the group key, it is possible to write an history describing how this element has been built from the group generator α. This history is linear since the combination of two elements of $\mathcal{G}$ into a third one never occurs.

Definition 2.9 *Given a bundle $\mathcal{C}$, a node $n \in \mathcal{N}_\mathcal{C}$, and an index i ($i \leq length(uns\_term(n))$, a* possible history *of (n, i) is a sequence $\langle (n_1, i_1), \ldots (n_m, i_m)\rangle$ such that:*

- $n_j \in \mathcal{N}_\mathcal{C}$ $(1 \leq j \leq m)$
- $term(n_1) = +t$ *and* $term(n_m) = -t'$
- $n_m = n$ *and* $i_m = i$
- $(n_{2j+1}, n_{2j+2}) \in \rightarrow_\mathcal{C}$ $(0 \leq j < m/2)$
- $(n_{2j}, n_{2j+1}) \in \Rightarrow_\mathcal{C}^+$ $(0 < j < m/2)$

We introduce a few more definitions about possible histories:

Definition 2.10 *Consider a bundle $\mathcal{C}$, an atom a, a node $n \in \mathcal{N}_\mathcal{C}$, an index i ($i \leq length(uns\_term(n))$, and let $\alpha_h = (n_1, i_1), \ldots (n_m, i_m)$ be a possible history of (n, i).*

(1) $\alpha_h(j) = (n_j, i_j)$;
(2) $\langle \alpha_h, j\rangle = uns\_term(n_j)(i_j) \in \mathsf{G}$; *for the simplicity of the further definitions, the element $\langle \alpha_h, 0\rangle$ is defined as α;*
(3) $node(\alpha_h(j)) = n_j$;
(4) $P(\alpha_h(j)) = p : \langle \alpha_h, j\rangle = \mathbf{exp}(\langle \alpha_h, j-1\rangle, p)$ $(0 < j \leq m)$
(5) $strand(\alpha_h(j)) = strand(n_j)$
(6) $Id(\alpha_h(j)) = M_k$ *where M_k is the user executing $strand(\alpha_h(j))$.*
(7) *a is* known *on $\alpha_j(k) = (n, i)$ iff $term(n) = +t$ (what is equivalent to say that k is odd) and $a \sqsubset P(\alpha_j(k))$;*

From this definition, $\alpha_h(j)$ is the j-th element of the sequence α_h, $\langle \alpha_h, j\rangle$ is the element of G exchanged at the j-th point of α_h, $node(\alpha_h(j))$ is the j-th node of α_h, $P(\alpha_h(j))$ is the value that has to be used for computing $\langle \alpha_h, j\rangle$ from $\langle \alpha_h, j-1\rangle$, $strand(\alpha_h(j))$ is the strand n_j belongs to, $Id(\alpha_h(j))$ is the identifier of the user executing $strand(\alpha_h(j))$, and a is said to be known on $\alpha_h(j)$ if it has to be used to compute $\langle \alpha_h, j\rangle$ from $\langle \alpha_h, j-1\rangle$. These notions are exemplified below.

Example 2.7 In Example 2.5, a possible history of $\langle s_2, 3\rangle(2) = \alpha^{r_1 r_3 K_{23}^2}$ is

$$\alpha_h = (\langle s_1, 1\rangle, 2), (\langle s_2, 1\rangle, 2), (\langle s_2, 2\rangle, 2), (\langle s_3, 1\rangle, 2), (\langle s_3, 2\rangle, 2), (\langle s_2, 3\rangle, 2)$$

$$\langle \alpha_h, 1\rangle = \alpha^{r_1}, \langle \alpha_h, 3\rangle = \alpha^{r_1 K_{23}}, \langle \alpha_h, 5\rangle = \alpha^{r_1 r_3 K_{23}^2}$$

$$P(\alpha_h(1)) = r_1, P(\alpha_h(2)) = 1, P(\alpha_h(5)) = r_3 K_{23}$$
$$strand(\alpha_h(1)) = s_1, strand(\alpha_h(2)) = s_2, Id(\alpha_h(6)) = M_2$$
$$K_{23} \text{ is known on } \alpha_h(3) \text{ and on } \alpha_h(5)$$

We can now define the class of protocols we consider.

Definition 2.11 *A* GDH-Protocol *on a group of n principals $\mathsf{M} = \{M_1, \ldots, M_n\}$ is a protocol the goal of which is to enable the sharing of a key $\alpha^{r_1 \ldots r_n}$ by the principals in M and the regular execution of which can be described through two elements:*

1. *a bundle $\mathcal{C}_{GDH}$ containing n strands $s_1 \ldots s_n$, M_i being the active principal for s_i. This part of the definition expresses how the GDH-Terms are exchanged.*
2. *n sequences $\alpha_1 \ldots \alpha_n$ of couples (n, i) where $n \in \mathcal{C}_{GDH}$, $i \leq length(uns\_term(n))$. These sequences express how the exchanged GDH-Terms are computed.*
   Let $(n_j^F, i_j^F) = \alpha_j(length(\alpha_j))$ and $\alpha_j^F = \langle \alpha_j, length(\alpha(j)) \rangle$.
   (a) *M_j computes the group key from α_j^F (so, $strand(n_j^F) = s_j$). Let $p_j^F \in \mathsf{P} : \mathbf{exp}(\alpha_j^F, p_j^F) = \alpha^{r_1 \ldots r_n}$*
   (b) *α_j is a possible history of (n_j^F, i_j^F)*
   (c) *$\langle \alpha_j, 2k+1 \rangle$ is computed from $\langle \alpha_j, 2k \rangle$ by $Id(\alpha_j(2k+1))$*
   We say that the atom a is locally known *in $\mathcal{C}_{GDH}$ iff $\exists j, k$ such that a is known on $\alpha_j(k)$ and $\forall\ l, m$, if a is known on $\alpha_l(m)$, then $strand(\alpha_j(k)) = strand(\alpha_l(m))$.*
   (d) *If $a \in \mathsf{R}$ is known on $\alpha_j(k)$ then it is locally known*
   (e) *For any $\alpha_j \neq \alpha_i$, there exists at least one index k such that the contribution r_i is known on $\alpha_j(k)$ and $strand(\alpha_j(k)) = s_i$.*
   (f) *If $a \in \mathsf{K}$ is known on $\alpha_j(k)$ then $a \in \mathsf{K}_{Id(\alpha_j(k))}$*
   (g) *If $a \in \mathsf{R}$ is known on $\alpha_j(k)$ and $a \sqsubset (p_l^F)$, then $strand(\alpha_j(k)) = s_l$*
   (h) *If $a \sqsubset p_j^F$ ($a \in \mathsf{K}$), then $a \in \mathsf{K}_j$*

Example 2.8 Our Ex-GDH protocol is an example of protocol respecting this definition and there is only one way to define α_1, α_2 and α_3 for this protocol:

$$\alpha_1 = \langle (\langle s_1, 1 \rangle, 1), (\langle s_2, 1 \rangle, 1), (\langle s_2, 2 \rangle, 1), (\langle s_3, 1 \rangle, 1), (\langle s_3, 2 \rangle, 1), (\langle s_2, 3 \rangle, 1) \rangle$$
$$\alpha_2 = \langle (\langle s_1, 1 \rangle, 2), (\langle s_2, 1 \rangle, 2), (\langle s_2, 2 \rangle, 2), (\langle s_3, 1 \rangle, 2), (\langle s_3, 2 \rangle, 2), (\langle s_2, 3 \rangle, 2) \rangle$$
$$\alpha_3 = \langle (\langle s_1, 1 \rangle, 2), (\langle s_2, 1 \rangle, 2), (\langle s_2, 2 \rangle, 3), (\langle s_3, 1 \rangle, 3) \rangle$$

The sequences $\alpha_1 \ldots \alpha_n$ express how the elements of G that will be used to compute the group key are built (points 2a, 2b and 2c). These

histories are used to define the notion of local knowledge: an atom is locally known if all points where it is known belong to the same strand. Typically, this will be the case for the random values generated during an execution of the protocol: since they are never communicated in a readable form and are not guessable, they cannot be used to compute elements of G on more than one strand (point 2d). We also impose that the contribution r_i to the group key is communicated by M_i on the element of G that will be used by the other group members to compute the group key (point 2e). These last two conditions notably impose that r_i must be generated by M_i and be kept secret. In point 2f, we say that the user M_i can only use keys he is supposed to know when he builds new elements of G. Point 2g expresses that the random values used by M_j to compute the group key are not known on any strand executed by an other group member, while point 2h expresses that M_j can only use keys he knows to compute the group key.

We will now introduce a few definitions and notations more before writing properties of GDH-Protocols.

Definition 2.12 *By default, we always refer to a GDH-protocol for a group* $\mathsf{M} = \{M_1 \cdots M_n\}$ *described through a GDH-Bundle* $\mathcal{C}_{GDH}$ *and through sequences* $\alpha_1 \ldots \alpha_n$ *. Let:*

(1) $C(M_j \rightarrow M_i) = \prod p_k : p_k = P(\alpha_i(k))$ *and* $Id(\alpha_i(k)) = M_j$ $(0 < k \leq length(\alpha_i))$*;* $C(M_j \rightarrow M_i)$ *represents the contribution that* M_j *gives to* α_i^F*;*
(2) $F_i = \mathbf{alphaexp}^{-1}(\alpha_i^F)$*;*
(3) $R = r_1 \ldots r_n$*;*
(4) $R_i = R \cdot (F_i)_{\mathsf{R}}^{-1} = (p_i^F)_{\mathsf{R}}$*;*
(5) $K_i = (F_i)_{\mathsf{K}}^{-1} = (p_i^F)_{\mathsf{K}}$.

Example 2.9 The table below indicates the value of $C(M_i \rightarrow M_j)$ for the Ex-GDH protocol in the line M_i of column M_j. The next tables indicate the value of F_i, R_i and K_i.

C	M_1	M_2	M_3
M_1	$\hat{r}_1$	r_1	r_1
M_2	$r_2 K_{23}$	K_{23}	r_2
M_3	$r_3 K_{13} K_{23}^{-1}$	$r_3 K_{23}$	1

$F_1 = \hat{r}_1 r_2 r_3 K_{13}$	$R_1 = r_1 (\hat{r}_1)^{-1}$	$K_1 = K_{13}^{-1}$
$F_2 = r_1 r_3 K_{23}^2$	$R_2 = r_2$	$K_2 = K_{23}^{-2}$
$F_3 = r_1 r_2$	$R_3 = r_3$	$K_3 = 1$

Definition 2.13 *Consider a GDH-Protocol. We say that an atom* a *is known on the strand* s_i *if there exist* j *and* k *such that* a *is known on*

$\alpha_j(k)$ and $strand(\alpha_j(k)) = s_i$. We also say that the product $p \in \mathsf{P}$ is known on the strand s_i if $\forall a \sqsubset p$, the atom a is known on s_i.

Similarly, we say that a is locally known on the strand s_i if there exist j and k such that a is known on $\alpha_j(k)$, $strand(\alpha_j(k)) = s_i$ and a is locally known in $\mathcal{C}_{GDH}$. Finally, we say that the product $p \in \mathsf{P}$ is locally known on the strand s_i if $\forall a \sqsubset p$, the atom a is locally known on s_i.

3. Properties of GDH-Protocols

We now define a few constitutive properties of GDH-Protocols. These properties express characteristics that GDH-Protocols must respect if they conform to their definition. They are considered in the absence of any attacker, and we will show in the next sections how they can be exploited in order to break security properties of such protocols.

It can be observed in the following paragraphs that we never precisely specify to which session of a protocol we refer: we simply state the corresponding group constitution when it is different from M. This is because we will always consider a single protocol execution for each specified group constitution. If, in a different context, a situation imposed us to consider several sessions of a protocol executed by the same group of users, we simply would need to add some supplementary references or indices in order to identify the strands to which we refer for the values local to specific sessions.

We now start our list of properties with two observations that will be used further.

Observation 2.2 *Let p_1, p_2 and p_3 be elements of P and a be an atom. If $p_1 = p_2 \cdot p_3$ and $a \sqsubset p_1$, then $a \sqsubset p_2$ or $a \sqsubset p_3$. Similarly, If $p_1 = p_2 \cdot p_3$ and $(p_1)_a = (p_2)_a$ then $a \not\sqsubset p_3$.*

Observation 2.3 *From the definition of F_j,*

(1) $(F_j)_{\mathsf{R}} = \prod_{i=1\ldots n} C_{\mathsf{R}}(M_i \rightarrow M_j)$
(2) $(F_j)_{\mathsf{K}} = \prod_{i=1\ldots n} C_{\mathsf{K}}(M_i \rightarrow M_j)$

This observation can be verified in Example 2.9. We can now write a first proposition about the value of $C_{\mathsf{R}}(M_i \rightarrow M_j)$ when $i \neq j$.

Proposition 2.4 *For any GDH-Protocol, if $1 \leq i, j \leq n$, $i \neq j$, then $C_{\mathsf{R}}(M_i \rightarrow M_j) = r_i$*

Proof. From Observation 2.3 and the definition of R_j, we can write

$$\prod_{i=1\ldots n} C_{\mathsf{R}}(M_i \rightarrow M_j) \cdot R_j = R \tag{1}$$

We can observe that $r_i \sqsubset R$. Furthermore, $r_i \not\sqsubset C_{\mathsf{R}}(M_k \rightarrow M_j)$ $(k \neq i)$ else $\exists l : r_i \sqsubset P(\alpha_j(l))$ and $Id(\alpha_j(l)) = M_k$ what is impossible given

points 2d and 2e of the definition of the GDH-Protocols. Finally, $r_i \not\sqsubset R_j$ given points 2e and 2g of the same definition. We can deduce from these remarks and from Observation 2.2 that $r_i \sqsubset C_{\mathsf{R}}(M_i \rightarrow M_j)$ and that $C_{r_i}(M_i \rightarrow M_j) = (R)_{r_i} = r_i$.

Let us now imagine that $C_{\mathsf{R}}(M_i \rightarrow M_j) = r_i \cdot r$. Then $r_i \not\sqsubset r$. Suppose $r_a \sqsubset r$. From Observation 2.2, $r_a \sqsubset C_{\mathsf{R}}(M_i \rightarrow M_j)$. Since r_a is known on s_i, it is locally known on s_i and is therefore not known on s_k $(k \neq i)$. So, $r_a \not\sqsubset C_{\mathsf{R}}(M_k \rightarrow M_j)$ $(k \neq i)$, $r_a \not\sqsubset R_j$ (from point 2g of Def. 2.11) and $r_a \not\sqsubset R$, what contradicts Observation 2.2 and equation (1). ∎

Concerning the value of $C_{\mathsf{R}}(M_i \rightarrow M_i)$, the following relation must be valid:

Proposition 2.5 *For any GDH-Protocol,* $C_{\mathsf{R}}(M_i \rightarrow M_i) = r_i \cdot R_i^{-1}$.

Proof. By definition, $R_i = R \cdot (F_i)_{\mathsf{R}}^{-1}$ and $R = \prod_{j=1\ldots n} r_j$. So, by successively exploiting Observation 2.3 and Proposition 2.4, we can write:

$$\begin{aligned} R_i &= \prod_{j=1\ldots n} r_j \cdot \left(\prod_{j=1\ldots n} C_{\mathsf{R}}(M_j \rightarrow M_i) \right)^{-1} \\ &= \prod_{j=1\ldots n} r_j \cdot \left(\prod_{j=1\ldots n,\ j\neq i} r_j \right)^{-1} \cdot C_{\mathsf{R}}(M_i \rightarrow M_i)^{-1} \\ &= r_i \cdot C_{\mathsf{R}}(M_i \rightarrow M_i)^{-1} \end{aligned}$$

∎

These two propositions can be checked for the Ex-GDH protocol in the tables of Example 2.9.

Having characterized the value of $C_{\mathsf{R}}(M_j \rightarrow M_i)$, we will now write two propositions concerning the value of $C_{\mathsf{K}}(M_j \rightarrow M_i)$.

Proposition 2.6 *For any GDH-Protocol, if* $C_{K_{jk}}(M_j \rightarrow M_i) = K_{jk}^a$ $(i \neq j, k)$ *then* $C_{K_{jk}}(M_k \rightarrow M_i) = K_{jk}^{-a}$.

Proof. By Observation 2.3, $\prod_{l=1\ldots n} C_{\mathsf{K}}(M_l \rightarrow M_i) \cdot K_i = 1$; so the sum of the powers of K_{jk} in the components of the left part of this equation must be null. But $K_{jk} \not\sqsubset K_i$ since $K_{jk} \notin \mathsf{K}_i$. Just as $K_{jk} \not\sqsubset C_{\mathsf{K}}(M_l \rightarrow M_i)$ $(l \neq j, k)$ since $K_{jk} \notin \mathsf{K}_l$. Therefore, K_{jk} can only be a subterm of $C_{\mathsf{K}}(M_j \rightarrow M_i)$ and of $C_{\mathsf{K}}(M_k \rightarrow M_i)$, and the powers of K_{jk} in these two contributions must be of the form a and $-a$ since their sum is null.

∎

Corollary 2.7 *For any GDH-protocol, if* $i \neq j$, $C_{\mathsf{K}}(M_i \rightarrow M_j) = \prod_{k=1\ldots n, k\neq i} C_{K_{ik}}(M_i \rightarrow M_j)$.

Proof. $C_{\mathsf{K}}(M_i \rightarrow M_j)$ can only contain keys of the form $K_{ix} \in \mathsf{K}_i$ since these are the only known on s_i. From Proposition 2.6, we can observe that $C_{K_{ii}}(M_i \rightarrow M_j) = 1$. Furthermore, $C_{K_{ik}}(M_i \rightarrow M_j) = 1$ when $k \neq 1 \ldots n$ since $C_{K_{ik}}(M_k \rightarrow M_i)$ is undefined (or can be considered as having value 1) when $M_k \notin \mathsf{M}$ ∎

Rather than considering the relations between values inside one session of a protocol, we would now like to write a proposition concerning the use of long-term keys in different sessions. To this effect, we introduce a substitution operator: if $p \in \mathsf{P}$ is such that $p_{\mathsf{R}} = 1$ and is a function of elements of a bundle corresponding to a session of a GDH-Protocol, $[M_i \backslash M_I : p]$ (where $M_i \in \mathsf{M}$ and $M_I \notin \mathsf{M}$) refers to the value of p that would have been computed in a session where the participants are the same except that M_i is substituted with M_I. More precisely:

Definition 2.14 *If* $p = \prod_j K_{ij}^{e_{ij}} \cdot K_x$ *where* $K_{ij} \not\sqsubset K_x$ $(\forall j)$ *then* $[M_i \backslash M_I : p] = \prod_j K_{Ij}^{e_{ij}} \cdot K_x$. *More generally, if* $S = \{M_{i_1}, \ldots M_{i_s}\}$, $[S \backslash M_I : p] = [M_{i_1} \backslash M_I : [(S \backslash \{M_{i_1}\}) \backslash M_I : p]]$.

Example 2.10 In the Ex-GDH protocol, $[M_1 \backslash M_I : C_{\mathsf{K}}(M_3 \rightarrow M_1)] = K_{I3} K_{23}^{-1}$ and $[\{M_1, M_2\} \backslash M_I : C_{\mathsf{K}}(M_3 \rightarrow M_1)] = K_{I3} K_{I3}^{-1} = 1$

As in the previous chapter, M_I denotes a user that is not a member of the group M and plays the role of the intruder. This user is however considered as a legitimate member of some other groups; $K_{Ij} \in (\mathsf{K}_I \cap \mathsf{K}_j)$ denoting a key shared by M_I and M_j.

We can now write a proposition relating the key part of the contribution of a honest member M_j, i.e. $C_{\mathsf{K}}(M_j \rightarrow M_i)$, with his contribution $[\mathsf{M}_s \backslash M_I : C_{\mathsf{K}}(M_j \rightarrow M_i)]$ in a session where a set of honest members $\mathsf{M}_s \subset \mathsf{M}$ has been replaced with the intruder. These two values are in fact equal, excepted that all occurrences of keys shared between M_j and users in M_s will be replaced by keys shared between M_j and M_I.

Proposition 2.8 *Let* $\mathsf{M}_s \subset \mathsf{M}$, $M_j \notin \mathsf{M}_s$. *Then* $C_{\mathsf{K}}(M_j \rightarrow M_i) = [\mathsf{M}_s \backslash M_I : C_{\mathsf{K}}(M_j \rightarrow M_i)] \cdot \prod_{M_k \in \mathsf{M}_s} C_{K_{jk}}(M_j \rightarrow M_i) \cdot \prod_{M_k \in \mathsf{M}_s} [\mathsf{M}_s \backslash M_I : C_{K_{jk}}^{-1}(M_j \rightarrow M_i)]$.

Proof. $C_{\mathsf{K}}(M_j \rightarrow M_i)$ is known on s_j, so it can be written as a product of keys of the form K_{jx}. A possible way to write $C_{\mathsf{K}}(M_j \rightarrow M_i)$ is therefore $\prod_{M_k \in \mathsf{M}_s} K_{jk}^{e_k} \cdot K_x$ where $K_{jk} \not\sqsubset K_x$ for all $M_k \in \mathsf{M}_s$ and K_x is a product of keys in K_j. Definition 2.14 now implies that $[\mathsf{M}_s \backslash M_I : C_{\mathsf{K}}(M_j \rightarrow M_i)] = \prod_{M_k \in \mathsf{M}_s} K_{jI}^{e_k} \cdot K_x$.

This proposition results from the fact that $K_{jk}^{e_k}$ can be written as $C_{K_{jk}}(M_j \rightarrow M_i)$ and that $K_{jI}^{e_k}$ can be written as $[M_k \backslash M_I : C_{K_{jk}}(M_j \rightarrow M_i)]$. ∎

Example 2.11 Consider the Ex-GDH protocol and $\mathsf{M}_s = \{M_1\}$. In this case, $C_{\mathsf{K}}(M_3 \rightarrow M_1) = K_{13}K_{23}^{-1}$, $[M_1 \backslash M_I : C_{\mathsf{K}}(M_3 \rightarrow M_1)] = K_{I3}K_{23}^{-1}$, $C_{K_{13}}(M_3 \rightarrow M_1) = K_{13}$ and $[M_1 \backslash M_I : C_{K_{13}}(M_3 \rightarrow M_1)] = K_{I3}$.

4. Expression of $R_i \cdot K_i$ as a Product of Contributions

We now describe how the properties of GDH-Protocols described in the previous section can be used to attack these protocols. The aim of the intruder will be to break the implicit key authentication property, i.e. obtain the key computed by any of the group members at the end of the protocol. This can be done for the group member M_i if the intruder is able to obtain two elements $g_1, g_2 \in \mathsf{G}$ such that $\mathbf{exp}(g_1, R_i \cdot K_i) = g_2$ and submit g_1 as the value M_i will use to compute the group key. These elements can be obtained by exploiting the services honest users are providing when they execute the protocol: when user M_i transforms $\alpha_j(k)$ into $\alpha_j(k+1)$ by exponentiating it with a product p of random values and keys, he allows the intruder to construct the pairs (g_1^p, g_2) and (g_1, g_2^p) from any pair (g_1, g_2) he already knows. Our goal in this section will be to show how any secret product $R_i \cdot K_i$ can be written as a product of contributions (which are product of services) offered by the users when they execute a GDH-Protocol. In the next section, we will examine how this expression of $R_i \cdot K_i$ can be used to build an attack against any GDH-Protocol.

In other words, from the previous chapter's point of view, we will show that, for certain attribution of the roles of the users in different sessions of the protocol, the linear system of Section 4.4.3 always has a number of solutions. In Section 5, we will show that it is always possible to undermine the IKA property by exploiting services as indicated in these solutions, what corresponds to the verification of the seven conditions of Section 4.4.4 of that chapter.

Before introducing our theorem, we write a lemma that indicates a way of writing K_i as a product of contributions in any GDH-Protocol.

Lemma 2.9 *Consider a GDH-Protocol executed by a group of users* $\mathsf{M} = \{M_1 \dots M_n\}$ *where* $n \geq 3$. *Let* M_i, M_j *and* M_k *be three different members of the group* M. *Let* S_j *and* S_k *be two disjoint sets of users such that* $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j$, $M_i \notin \mathsf{S}_k$ *and* $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$. *Then*

$$\begin{aligned} K_i^{-1} \;=\; & C_{\mathsf{K}}(M_i \rightarrow M_i) \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \rightarrow M_i)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}(M_l \rightarrow M_i)] \cdot \prod_{m \in 1 \dots n} K_{Im}^{e_m} \end{aligned}$$

Proof. We start from Observation 2.3 which gives us that

$$K_i^{-1} = \prod_{j=1\ldots n} C_{\mathsf{K}}(M_j \to M_i)$$

and prove the announced identity in four steps: in the first three ones we prove particular identities, while in the last one we show the lemma by gathering these identities and the previous observation.

1) We prove that $\prod_{M_j \in \mathsf{M}\setminus\{M_i\}} C_{\mathsf{K}}(M_j \to M_i) = \prod_{M_j \in \mathsf{M}\setminus\{M_i\}} C_{K_{ij}}(M_j \to M_i)$.
From Corollary 2.7, $C_{\mathsf{K}}(M_j \to M_i) = \prod_{M_m \in \mathsf{M}\setminus\{M_j\}} C_{K_{jm}}(M_j \to M_i)$ and, from Proposition 2.6, $C_{K_{jm}}(M_j \to M_i) = C^{-1}_{K_{jm}}(M_m \to M_i)$.

So, the only terms that are not inverted in $\prod_{M_j \in \mathsf{M}\setminus\{M_i\}} \prod_{M_m \in \mathsf{M}\setminus\{M_j\}} C_{K_{jm}}(M_j \to M_i)$ are those of the form $C_{K_{ji}}(M_j \to M_i)$ (for $M_j \in \mathsf{M}\setminus\{M_i\}$), what proves our identity.

2) The second identity we prove is $\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] = \prod_{M_l \in \mathsf{S}_k} C_{K_{li}}(M_l \to M_i) \cdot \prod_{l \in 1\ldots n} K_{Il}^{e_l}$
Let $M_l \in \mathsf{S}_k$. From Corollary 2.7, $[\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] = [\mathsf{S}_j \to M_I : \prod_{M_m \in \mathsf{M}\setminus\{M_l\}} C_{K_{lm}}(M_l \to M_i)]$. If $M_m \in \mathsf{S}_j$, then $[\mathsf{S}_j \to M_I : C_{K_{lm}}(M_l \to M_i)]$ is a term of the form $K_{lI}^{e_l}$, and is therefore known by M_I. So, $[\mathsf{S}_j \to M_I : \prod_{M_m \in \mathsf{M}\setminus\{M_l\}} C_{K_{lm}}(M_l \to M_i)] = [\mathsf{S}_j \to M_I : \prod_{m \in \mathsf{S}_k \cup \{M_i\}\setminus\{M_l\}} C_{K_{lm}}(M_l \to M_i)] \cdot K_{lI}^{e_l}$.

Finally, if we simplify the terms in

$$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \to M_I : \prod_{M_m \in \mathsf{S}_k \cup \{M_i\}\setminus\{M_l\}} C_{K_{lm}}(M_l \to M_i)]$$

by using Proposition 2.6 as in the previous paragraph, we can observe that the only remaining terms are those of the form $C_{K_{li}}(M_l \to M_i)$, what proves our identity.

3) Similarly $\prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \to M_I : C_{\mathsf{K}}(M_l \to M_i)] = \prod_{M_l \in \mathsf{S}_j} C_{K_{li}}(M_l \to M_i) \cdot \prod_{l \in 1\ldots n} K_{Il}^{e_l}$

4) The first point of our proof shows that $K_i^{-1} = C_{\mathsf{K}}(M_i \to M_i) \cdot \prod_{M_j \in \mathsf{M}\setminus\{M_i\}} C_{K_{ij}}(M_j \to M_i)$. The result of the lemma derives from the splitting of that term into $\prod_{M_l \in \mathsf{S}_j} C_{K_{il}}(M_l \to M_i) \cdot \prod_{M_l \in \mathsf{S}_k} C_{K_{il}}(M_l \to M_i)$, and from the exploitation of the identities of the second and third parts of the proof.

∎

Using this lemma, we prove the announced theorem.

Theorem 2.10 *For any GDH-Protocol executed by a group of users* $\mathsf{M} = \{M_1 \dots M_n\}$ *where* $n \geq 3$*, it is possible to write any secret* $R_i \cdot K_i$ *as a product of contributions* $C(M_j \to M_k)$ $(M_j, M_k \in \mathsf{M} \cup \{M_I\})$ *and of keys known by* M_I*.*

Proof. Let S_j and S_k be two disjoint sets of users such that $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j$, $M_i \notin \mathsf{S}_k$ and $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$.

We split our proof into three parts: in the first one we treat the product R_i, in the second one we treat the product K_i, while in the last one we gather the terms of the first two parts.

1) Rewriting R_i

From Propositions 2.4 and 2.5 , we know that

$$R_i = C_{\mathsf{R}}(M_i \to M_j) \cdot C_{\mathsf{R}}(M_i \to M_i)^{-1}$$

But $C_{\mathsf{R}}(M_i \to M_j) = C(M_i \to M_j) \cdot C_{\mathsf{K}}^{-1}(M_i \to M_j)$, and, from Proposition 2.8,

$$\begin{aligned} C_{\mathsf{K}}^{-1}(M_i \to M_j) &= [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \to M_j)] \cdot \prod_{M_l \in \mathsf{S}_j} C_{K_{il}}^{-1}(M_i \to M_j) \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_j \backslash M_I : C_{K_{il}}(M_i \to M_j)] \end{aligned}$$

The last term of this equation is a product of keys M_I knows. Furthermore,

$$[\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \to M_j)] = [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \to M_j) \cdot C_{\mathsf{R}}(M_i \to M_j)]$$

and, from Proposition 2.4,

$$[\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}(M_i \to M_j)] = [\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}(M_i \to M_k)]$$

Finally,

$$[\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}(M_i \to M_k)] = [\mathsf{S}_j \backslash M_I : C(M_i \to M_k) \cdot C_{\mathsf{K}}^{-1}(M_i \to M_k)]$$

So, if we define

$$\begin{aligned} PR &= C(M_i \to M_j) \cdot [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)] \cdot \\ &\quad [\mathsf{S}_j \backslash M_I : \prod_{M_l \in \mathsf{S}_j} C_{K_{il}}(M_i \to M_j)] \end{aligned}$$

that is a product of contributions and of keys the intruder knows, we can write

$$\begin{aligned} R_i &= PR \cdot C_{\mathsf{R}}^{-1}(M_i \to M_i) \cdot [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \to M_k)] \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_j} C_{K_{il}}^{-1}(M_i \to M_j) \end{aligned}$$

2) Rewriting K_i

From Lemma 2.9, we know that

$$\begin{aligned} K_i^{-1} &= C_{\mathsf{K}}(M_i \to M_i) \cdot \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] \cdot \prod_{m \in \mathsf{M}} K_{Im}^{e_m} \end{aligned}$$

But

$$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] = \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C(M_l \to M_i) \cdot C_{\mathsf{R}}^{-1}(M_l \to M_i)]$$

and, from Proposition 2.4,

$$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}^{-1}(M_l \to M_i)] = \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}^{-1}(M_l \to M_k)]$$

Finally,

$$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{R}}^{-1}(M_l \to M_k)] = \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \to M_k) \cdot C_{\mathsf{K}}(M_l \to M_k)]$$

Symmetrically, it can be shown that

$$\prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}(M_l \to M_i)] = \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C(M_l \to M_i) \cdot C^{-1}(M_l \to M_j) \cdot C_{\mathsf{K}}(M_l \to M_j)]$$

Finally, if we define

$$\begin{aligned} PK &= \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C(M_l \to M_i) \cdot C^{-1}(M_l \to M_k)] \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C(M_l \to M_i) \cdot C^{-1}(M_l \to M_j)] \cdot \prod_{M_m \in \mathsf{M}} K_{Im}^{e_m} \end{aligned}$$

we have shown that

$$\begin{aligned} K_i &= PK \cdot C_{\mathsf{K}}^{-1}(M_i \to M_i) \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_k)] \cdot \\ &\quad \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_j)] \end{aligned}$$

3) Gathering of the terms

If we gather the terms of the first two points, we can see that

$$\begin{aligned} R_i \cdot K_i \;=\; & PR \cdot PK \cdot \\ & C_{\mathsf{R}}^{-1}(M_i \to M_i) \cdot C_{\mathsf{K}}^{-1}(M_i \to M_i) \cdot \\ & [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \to M_k)] \cdot \prod_{M_l \in \mathsf{S}_j} C_{K_{il}}^{-1}(M_i \to M_j) \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_j)] \end{aligned}$$

PR and PK are products of terms of the form $C(M_l \to M_m)$ and of keys M_I knows. Furthermore, $C_{\mathsf{R}}^{-1}(M_i \to M_i) \cdot C_{\mathsf{K}}^{-1}(M_i \to M_i) = C^{-1}(M_i \to M_i)$; so our theorem is proved if we can prove that the last four terms of the previous equation can be rewritten as a product of terms of the form $C(M_l \to M_m)$ and of keys M_I knows.

But

$$\begin{aligned} & [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \to M_k)] \cdot \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_k)] = \\ & \prod_{M_l \in \mathsf{M} \backslash \mathsf{S}_j} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \to M_k)] = \\ & [\mathsf{S}_j \backslash M_I : (F_k)_{\mathsf{K}}^{-1}] \cdot \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}(M_l \to M_k)] \end{aligned}$$

The last equality comes from the use of Observation 2.3 and these last two terms are products of keys M_I knows since $M_k \in \mathsf{S}_j$.

On the other side,

$$\begin{aligned} & \prod_{M_l \in \mathsf{S}_j} C_{K_{il}}^{-1}(M_i \to M_j) = \\ & \prod_{M_l \in \mathsf{M} \backslash \{M_i\}} C_{K_{il}}^{-1}(M_i \to M_j) \cdot \prod_{M_l \in \mathsf{S}_k} C_{K_{il}}(M_i \to M_j) = \\ & C_{\mathsf{K}}^{-1}(M_i \to M_j) \cdot \prod_{M_l \in \mathsf{S}_k} C_{K_{il}}(M_i \to M_j) = \\ & [\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}(M_i \to M_j)] \cdot \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_k \backslash M_I : C_{K_{il}}(M_i \to M_j)] \end{aligned}$$

The first equality exploits the definition of S_j and S_k, the second one exploits Corollary 2.7, and the last one exploits Proposition 2.8. We can observe that $\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_k \backslash M_I : C_{K_{il}}(M_i \to M_j)]$ is a product of keys known by M_I and that $[\mathsf{S}_k \backslash M_I : C_{\mathsf{K}}(M_i \to M_j)] \cdot \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I :$

$C_{\mathsf{K}}^{-1}(M_l \rightarrow M_j)]$ is also known by M_I for the same reason $[\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_i \rightarrow M_k)] \cdot \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C_{\mathsf{K}}^{-1}(M_l \rightarrow M_k)]$ was.

Finally, we have shown that

$$\begin{aligned} R_i \cdot K_i \;=\; & C^{-1}(M_i \rightarrow M_i) \cdot C(M_i \rightarrow M_j) \cdot \\ & [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \rightarrow M_j) \cdot C(M_i \rightarrow M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot (M_l \rightarrow M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_j)] \cdot \prod_{M_l \in \mathsf{M}} K_{Il}^{e_l} \end{aligned}$$

■

We illustrate this theorem by expressing $R_2 \cdot K_2$ in the case of the Ex-GDH protocol.

Example 2.12 In the Ex-GDH protocol, we choose $i = 2$, $j = 3$ and $k = 1$. Then $\mathsf{S}_j = \{M_1\}$ and $\mathsf{S}_k = \{M_3\}$. Our theorem says:

$$\begin{aligned} R_2 \cdot K_2 \;=\; & C^{-1}(M_2 \rightarrow M_2) \cdot C(M_2 \rightarrow M_3) \cdot \\ & [M_1 \backslash M_I : C^{-1}(M_2 \rightarrow M_3) \cdot C(M_2 \rightarrow M_1)] \cdot \\ & [M_1 \backslash M_I : C^{-1}(M_3 \rightarrow M_2) \cdot C(M_3 \rightarrow M_1)] \cdot \\ & [M_3 \backslash M_I : C^{-1}(M_1 \rightarrow M_2) \cdot C(M_1 \rightarrow M_3)] \cdot K_{I3}^{-1} \end{aligned}$$

It can effectively be verified by substitution that

$$\begin{aligned} r_2 \cdot K_{23}^{-2} \;=\; & K_{23}^{-1} \cdot r_2 \\ & (r_2')^{-1} \cdot r_2' K_{23} \\ & (r_3' K_{23})^{-1} \cdot r_3' K_{I3} K_{23}^{-1} \\ & (r_1'')^{-1} \cdot r_1'' \cdot K_{I3}^{-1} \end{aligned}$$

In order to be able to break the IKA property for any GDH-Protocol, we still need to show that it is always possible for the intruder to obtain a pair of the form (g_1, g_1^x) where x is the product of contributions at the end of the previous theorem, and to replace the value M_i will use to compute the group key with g_1. This is what we will examine in the next section.

5. Building Attacks Against GDH-Protocols

We now describe how the theorem of the previous section can be used to systematically build attacks against any GDH-Protocol. The exploitation of this theorem implies the construction of pairs of elements $(g_1, g_2) \in \mathsf{G} \times \mathsf{G}$ such that g_2 is equal to g_1 exponentiated with a product of elements of the form $C(M_i \rightarrow M_j)$.

The construction of such pairs can be done when the restrictions we draw in the next propositions are respected.

5.1. Building Simple Pairs of Elements of G

In this subsection, we will show how it is possible to build pairs of elements g_1, g_2 of G such that g_2 is equal to $g_1^{C(M_i \to M_j)}$ for any i and j (Proposition 2.11) or to $g_1^{C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$ for any i, j and k (Proposition 2.12). These propositions will be useful to understand how more complex pairs can be built, which will be done in the next subsection.

Proposition 2.11 *For any session of a GDH-Protocol executed by a group of users* M*, an active attacker can obtain* $g = \alpha^{C(M_i \to M_j)} \in \mathsf{G}$ *where* $M_i, M_j \in \mathsf{M}$*.*

Proof. Consider a session of the considered protocol executed by the group of users M and the history $\alpha_j = \langle (n_1, i_1), \ldots, (n_m, i_m) \rangle$ for that protocol. If we initialize g to α, Algorithm 3 gives the intruder the element required.

Algorithm 3 Defines a strand s_I which, when executed together with s_i, provides $g = \alpha^{C(M_i \to M_j)}$

---

for $k := 1$ to $length(s_i)$ **do**
  if $\exists t : term(\langle s_i, k \rangle) = +t$ **then**
    $term(\langle s_I, k \rangle) := -t$
    if $\exists x : \langle s_i, k \rangle = node(\alpha_j(x))$ **then**
      $g := \langle \alpha_j, x \rangle$
    end if
  else
    $t :=$ sequence of $length(term(\langle s_i, k \rangle))$ random elements of G
    if $\exists x : \langle s_i, k \rangle = node(\alpha_j(x))$ **then**
      $t(i_x) := g$
    end if
    $term(\langle s_I, k \rangle) = +t$
  end if
end for

This algorithm may be justified as follows. Let s_i be a strand that corresponds to M_i's role in an execution of the considered protocol by the group M. We proceed by constructing a strand s_I perfectly matching s_i, i.e. a strand such that $term(\langle s_i, x \rangle) = -term(\langle s_I, x \rangle)$. So, by executing this strand, the intruder will have a conversation with M_i at the end of which M_i will have completed his role in the considered session of the

protocol without interacting with any other member of M. Furthermore, at the end of this session, the variable g has the desired value.

The strand s_I is constructed by following s_i from its first to its last node:

- when a term is emitted on a node of s_i, we define a matching node on s_I intended to receive it;
- when a term has to be received on some node of s_i, we define a matching node on s_I from which a term of the appropriate length is emitted.

The required element of G is obtained as follows:

- when a term containing an element of α_j is emitted on the current node of s_i, the variable g is updated to its value;
- when a term containing an element of α_j has to be received on the current node of s_i, we set this element to g.

The correctness of the algorithm relies on two observations:

- $node(\alpha_j(x)) \prec node(\alpha_j(x+1))$
- if $node(\alpha_j(x))$ belongs to s_i and $term(node(\alpha_j(x))) = -t$, then $node(\alpha_j(x)) \Rightarrow^+ node(\alpha_j(x+1))$ and $term(node(\alpha_j(x))) = +t'$.

These observations allow us to write the following invariant that is valid at the end of each " **if** $\exists x : \langle s_i, k\rangle = node(\alpha_j(x))$" clause:

$$g = \alpha^{\prod p_z} : p_z = P(\alpha_j(z)) \text{ and } Id(\alpha_j(z)) = M_i, (0 < z \leq x)$$

Therefore, successively considering all nodes of s_i guarantees us that

$$g = \alpha^{\prod p_z} : p_z = P(\alpha_j(z)) \text{ and } Id(\alpha_j(z)) = M_i, (0 < z \leq length(\alpha_j))$$

at the end of the execution, which shows that $g = \alpha^{C(M_i \to M_j)}$ by definition of $C(M_i \to M_j)$. ∎

Example 2.13 We can apply our algorithm in order to obtain $g = \alpha^{C(M_2 \to M_1)}$ for our Ex-GDH protocol. For the record, we represented a typical run of this protocol in Fig. 2.2.

For that protocol,

$$\alpha_1 = \langle(\langle s_1, 1\rangle, 1), (\langle s_2, 1\rangle, 1), (\langle s_2, 2\rangle, 1), (\langle s_3, 1\rangle, 1), (\langle s_3, 2\rangle, 1), (\langle s_1, 3\rangle, 1)\rangle$$

where s_1, s_2 and s_3 are executed by M_1, M_2 and M_3 respectively.

Our algorithm will successively consider all nodes of s_2 in order to build s_I, the index k indicating which node of s_i is examined.

$k = 1$ $term(\langle s_2, 1\rangle)$ is negative, so we define $t := \langle \alpha^r, \alpha^r\rangle$ (where we chose the random values to be α^r). On the next test, the choice $x = 2$ makes the condition *true* so we redefine t as $t := \langle \alpha, \alpha^r\rangle$, and state $term(\langle s_I, 1\rangle) := +t$.

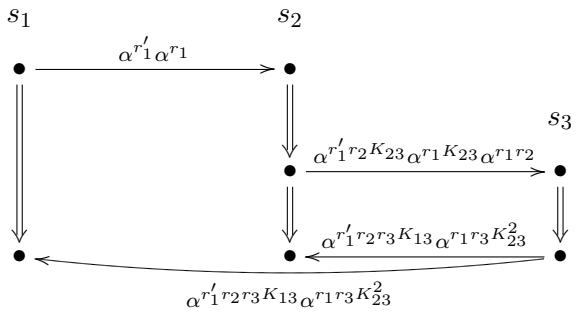


FIGURE 2.2. A run of the Ex-GDH protocol

$k = 2$ $term(\langle s_2, 2\rangle)$ is positive, so we define $term(\langle s_I, 2\rangle) := -t$ where $t = \langle \alpha^{r_2 K_{23}}, \alpha^{r K_{23}}, \alpha^{r r_2}\rangle$. Since the choice $x = 3$ matches the next **if** clause, we update the value of g to $\alpha^{r_2 K_{23}}$.

$k = 3$ $term(\langle s_2, 3\rangle)$ is negative, so we anew define $t := \langle \alpha^r, \alpha^r\rangle$. The node $\langle s_2, 3\rangle$ is not part of α_1, so the next **if** clause cannot be satisfied, $term(\langle s_I, 2\rangle)$ is set to $+t$, and the algorithm stops.

We can check that $g = \alpha^{r_2 K_{23}} = \alpha^{C(M_2 \to M_1)}$ as expected. s_2 and s_I are represented in Fig. 2.3.

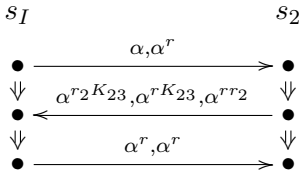


FIGURE 2.3. Representation of s_I and s_i

The algorithm we just proposed (and those we will propose further) proceeds by following strands rather than histories. The required value of G could also have been obtained by going through α_j and by checking whether the current node was on s_i: the corresponding algorithm would probably have been more natural, but we preferred the solution we proposed because it emphasizes the fact that we only need to interact with one strand independently of those corresponding to other roles in the protocol execution.

If we look at Theorem 2.10, we can observe that we are more generally interested in pairs (g_1, g_2) such that $g_2 = g_1^{C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$. We now examine how such pairs can be obtained.

Proposition 2.12 *For any session of a GDH-Protocol executed by a group of users* M *of cardinality* n*, an active attacker can obtain a pair* (g_1, g_2) *of elements of* G *such that* $g_2 = g_1^{C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$.

Proof. Consider a session of the considered protocol executed by the members of the group M. If we initialize g_1 and g_2 to α, Algorithm 4 gives the intruder a pair (g_1, g_2) of the desired form.

Algorithm 4 Defines a strand s_I which, when executed together with s_i, provides a pair (g_1, g_2) such that $g_2 = g_1^{C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$ $(M_j \neq M_k)$ if the precondition $g_1 = g_2$ is verified.

---

for $z := 1$ to $length(s_i)$ **do**
  if $\exists t : term(\langle s_i, z \rangle) = +t$ **then**
    $term(\langle s_I, z \rangle) := -t$
    if $\exists x : \langle s_i, z \rangle = node(\alpha_j(x))$ and $\alpha_j(x) \neq \alpha_k(x)$ **then**
      $g_1 := \langle \alpha_j, x \rangle$
    end if
    if $\exists y : \langle s_i, z \rangle = node(\alpha_k(y))$ and $\alpha_j(y) \neq \alpha_k(y)$ **then**
      $g_2 := \langle \alpha_k, y \rangle$
    end if
  else
    $t :=$ sequence of $length(term(\langle s_i, z \rangle))$ random elements of G
    if $\exists x : \langle s_i, z \rangle = node(\alpha_j(x))$ and $\alpha_j(x+1) \neq \alpha_k(x+1)$ **then**
      $t(i_x) := g_1$
    end if
    if $\exists y : \langle s_i, z \rangle = node(\alpha_k(y))$ and $\alpha_j(y+1) \neq \alpha_k(y+1)$ **then**
      $t(i_y) := g_2$
    end if
    $term(\langle s_I, z \rangle) = +t$
  end if
end for

The principle of this algorithm is similar to the previous one: we go through the strand s_i and construct a matching strand s_I, while collecting $\alpha^{C(M_i \to M_j)}$ into g_1 and $\alpha^{C(M_i \to M_k)}$ into g_2. The main difference between these two algorithms is that we consider the update of two values (i.e. g_1 and g_2) at each step of the **for** loop rather than simply one (i.e. g). There is one case where these updates may interfere: if $\alpha_j(x) = \alpha_k(y)$ for certain value of z. However, in this case, item 2c of Definition 2.11 guarantees us that in this case $x = y$ and that these two histories have all previous points in common and, for that reason, the variables g_1 and g_2 do not need to be updated to distinct values. ∎

Example 2.14 We apply Algorithm 4 in order to obtain a pair (g_1, g_2) such that $g_2 = g_1^{C^{-1}(M_2 \to M_2) \cdot C(M_2 \to M_3)}$ in our Ex-GDH protocol. For that protocol,

$$\alpha_2 = \langle(\langle s_1, 1\rangle, 2), (\langle s_2, 1\rangle, 2), (\langle s_2, 2\rangle, 2), (\langle s_3, 1\rangle, 2), (\langle s_3, 2\rangle, 2), (\langle s_2, 3\rangle, 2)\rangle$$

$$\alpha_3 = \langle(\langle s_1, 1\rangle, 2), (\langle s_2, 1\rangle, 2), (\langle s_2, 2\rangle, 3), (\langle s_3, 1\rangle, 3)\rangle$$

where s_1, s_2 and s_3 are executed by M_1, M_2 and M_3 respectively.

Our algorithm anew successively considers all nodes of s_2 in order to build s_I, the index k indicating which node of s_i is examined.

- $k = 1$ $term(\langle s_2, 1\rangle)$ is negative, so we define $t := \langle \alpha^r, \alpha^r \rangle$ (where we choose the random values to be α^r). On the next tests, the choices $x = 2$ and $y = 2$ make the conditions *true* so we redefine t as $t := \langle \alpha^r, \alpha \rangle$, and state $term(\langle s_I, 1\rangle) := +t$.
- $k = 2$ $term(\langle s_2, 2\rangle)$ is positive, so we define $term(\langle s_I, 2\rangle) := -t$ where $t = \langle \alpha^{r r_2 K_{23}}, \alpha^{K_{23}}, \alpha^{r_2} \rangle$. Since the choices $x = 3$ and $y = 3$ match the next two **if** clauses, we update the value of g_1 to $\alpha^{K_{23}}$ and of g_2 to α^{r_2}.
- $k = 3$ $term(\langle s_2, 3\rangle)$ is negative, so we anew define $t := \langle \alpha^r, \alpha^r \rangle$. The choice $x = 6$ matches the next test, so we redefine t as $t := \langle \alpha^r, \alpha^{K_{23}} \rangle$. The node $\langle s_2, 3\rangle$ is not part of α_1, so the next **if** clause cannot be satisfied. We finally state $term(\langle s_I, 3\rangle) := +t$ and the algorithm stops.

We can easily verify that $g_2 = g_1^{r_2 K_{23}^{-1}} = g_1^{C^{-1}(M_2 \to M_2) \cdot C(M_2 \to M_3)}$ as expected. s_2 and s_I are represented in Fig. 2.4.

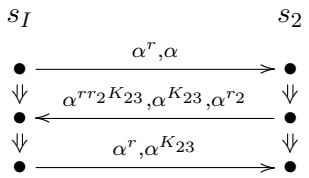


FIGURE 2.4. Representation of s_I and s_i

We are now able to obtain pairs (g_1, g_2) of elements of G such that $g_2 = g_1^{C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$ for any i, j and k by using Algorithm 4. If we look at the pairs we would like to obtain in order to exploit Theorem 2.10, we remark that we need a more general proposition: we should be able to transform a pair (g_1, g_2) such that $g_2 = g_1^p$ into a pair of the form (g_1, g_2) such that $g_2 = g_1^{p \cdot C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$.

5.2. Combining Pairs of Elements of G

We now present a number of sufficient conditions making it possible to transform a pair (g_1, g_2) such that $g_2 = g_1^p$ into a pair of the form (g_1, g_2) such that $g_2 = g_1^{p \cdot C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)}$. These conditions roughly correspond to a transposition of the seven conditions expressed in Section 4.4.4.

We start by exemplifying the problems we are confronted to.

Example 2.15 We introduce a new protocol that we call *Tri-GDH*. This protocol can be defined through three strands and three histories:

$$\begin{aligned}
s_1 &= \langle +\alpha^{r_1}, -\alpha^{r_3}, +\alpha^{r_1 r_2 K_{12}}, -\alpha^{r_2 r_3 K_{13}} \rangle \\
s_2 &= \langle +\alpha^{r_2}, -\alpha^{r_1}, +\alpha^{r_1 r_2 K_{23}}, -\alpha^{r_1 r_2 K_{12}} \rangle \\
s_3 &= \langle +\alpha^{r_3}, -\alpha^{r_2}, +\alpha^{r_2 r_3 K_{13}}, -\alpha^{r_1 r_2 K_{23}} \rangle \\
\alpha_1 &= \langle (\langle s_2, 1\rangle, 1), (\langle s_3, 2\rangle, 1), (\langle s_3, 3\rangle, 1), (\langle s_1, 4\rangle, 1) \rangle \\
\alpha_2 &= \langle (\langle s_3, 1\rangle, 1), (\langle s_1, 2\rangle, 1), (\langle s_1, 3\rangle, 1), (\langle s_2, 4\rangle, 1) \rangle \\
\alpha_3 &= \langle (\langle s_1, 1\rangle, 1), (\langle s_2, 2\rangle, 1), (\langle s_2, 3\rangle, 1), (\langle s_3, 4\rangle, 1) \rangle
\end{aligned}$$

A run of this protocol is represented in Fig. 2.5. The three central messages are exchanged first, while the three external are computed from those just received.

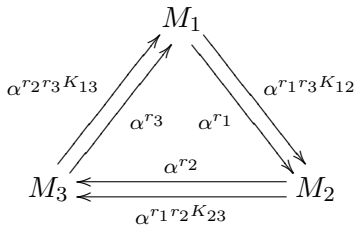


FIGURE 2.5. A run of the Tri-GDH protocol

An application of Theorem 2.10 for this protocol with $i = 1$, $j = 2$ and $k = 3$ gives:

$$\begin{aligned}
r_1 \cdot K_{13}^{-1} &= 1 \cdot r_1 K_{12} \cdot (r'_1 K_{12})^{-1} \cdot r'_1 \cdot \\
&\quad r'^{-1}_2 \cdot r'_2 K_{2I} \cdot (r''_3 K_{13})^{-1} \cdot r''_3 \cdot K_{2I}^{-1}
\end{aligned}$$

where r_i, r'_i, r''_i represent random values generated during three sessions of the protocol (the participants of these sessions being respectively $\{M_1, M_2, M_3\}$, $\{M_1, M_2, M_I\}$ and $\{M_1, M_I, M_3\}$).

Among these contributions we may consider r'_1, r'^{-1}_2 and r''_3: these three services are provided as first elements of histories. If we apply Algorithm 3 on the first two contributions (ignoring the negative exponent

of r_2'), we obtain the values $\alpha^{r_1'}$ and $\alpha^{r_2'}$. These values can be combined to obtain the pairs $(\alpha^{r_2'}, \alpha^{r_1'})$ and $(\alpha^{r_1'}, \alpha^{r_2'})$. This is the only solution we have to obtain pairs of that form since these services are provided independently of any input value. Now, we would like to transform that pair into a pair of the form $(\alpha^{r_2'}, \alpha^{r_1' r_3''})$. However, this is not so obvious since the service r_3'' is also provided independently of any input and the only way we can take this service into account is by using the value $\alpha^{r_3''}$ sent by M_3.

Guided by this example, we can more generally observe that we are not usually able to combine two contributions containing initial parts of histories into one pair of elements of G if we have to exploit these contributions in the same direction (i.e. if their powers have the same sign).

This restriction on the use of the starting point of histories is however not sufficient as we will see in the following example.

Example 2.16 Suppose we need to collect four services s_1, s_2, s_3 and s_4 in order to obtain a pair of the form $(\alpha^{s_2 s_3}, \alpha^{s_1 s_4})$ and that s_3 and s_4 are starting services. Suppose furthermore that s_1 and s_3 are provided by M_1 while s_2 and s_4 are provided by M_2. Finally, suppose that M_1 provides s_1 before providing s_3 and that M_2 provides s_2 before providing s_4. Such a scenario is represented in Fig. 2.6. We now try to obtain the

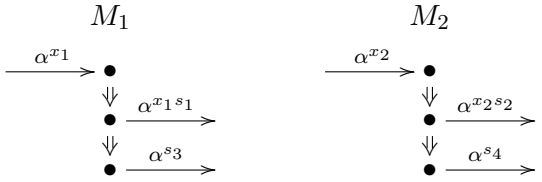


FIGURE 2.6. Representation of the roles of M_1 and M_2

desired pair. Since s_3 is provided independently of any input value, we have to exploit this service before exploiting s_2. However, the same thing is true for the s_4 service that we have to exploit before s_1.

These two constraints are unfortunately incompatible since s_1 is provided before s_3 and s_2 before s_4.

This last example shows us that we are not able to exploit services provided on two strands if we have to exploit starting services further on the same strands.

Besides starting services, there is another kind of problematic services we have to consider: splitting services.

Example 2.17 Consider our Ex-GDH protocol. The use of Theorem 2.10 proposed in Example 2.12 gave us the following identity:

$$\begin{aligned} R_2 \cdot K_2 \;=\; & C^{-1}(M_2 \to M_2) \cdot C(M_2 \to M_3) \cdot \\ & [M_1 \backslash M_I : C^{-1}(M_2 \to M_3) \cdot C(M_2 \to M_1)] \cdot \\ & [M_1 \backslash M_I : C^{-1}(M_3 \to M_2) \cdot C(M_3 \to M_1)] \cdot \\ & [M_3 \backslash M_I : C^{-1}(M_1 \to M_2) \cdot C(M_1 \to M_3)] \cdot K_{I3}^{-1} \end{aligned}$$

We now examine the product of contributions on the first line of this equation: $C^{-1}(M_2 \to M_2) \cdot C(M_2 \to M_3)$. If we apply Algorithm 4 to obtain a pair (g_1, g_2) such that $g_2 = g_1^{C^{-1}(M_2 \to M_2) \cdot C(M_2 \to M_3)}$, we may observe that when $z = 1$, $t(2)$ is successively affected to g_1 and g_2. This is due to the fact that the first two elements of α_2 and α_3 are identical. On the next step, when $z = 2$, g_1 and g_2 are affected to the outputs of services offered by M_2, services that are applied on one single value. As a result of this, we are not able to transform a given initial pair (g_1, g_2) into $(g_1^{C^{-1}(M_2 \to M_2)}, g_2^{C(M_2 \to M_3)})$ since our algorithm only provides $(g_2^{C^{-1}(M_2 \to M_2)}, g_2^{C(M_2 \to M_3)})$ (for any chosen value initial of g_2).

Having informally indicated which kind of situations may raise problems in the reconstruction of pairs of elements of G, we now describe sufficient conditions for the possibility of building the pairs of the form suggested in Theorem 2.10. These conditions are specified through the three notions we now define.

Definition 2.15 *Consider a GDH-Protocol with n participants and let $\alpha_1, \ldots, \alpha_n$ be the n histories given in the definition of this protocol. We define* $\mathrm{start}(M_i)$ *as* $Id(\alpha_i(1))$.

We say that the product of contributions $\prod_{i \in \mathcal{I}} C^{e_i}(M_{j_i} \to M_{k_i})$ (with $\mathcal{I}$ a set of indices, $e_i \in \{-1, 1\}$, $1 \leq j_i, k_i \leq n$) contains x start^+ *(resp.* start^-*) if there exist x indices in $\mathcal{I}$ such that $e_i = 1$ (resp. $e_i = -1$) and $start(M_{k_i}) = M_{j_i}$.*

By extension, we say that $\prod_{i \in \mathcal{I}} C^{e_i}(M_{j_i} \to M_{k_i})$ contains x starts *(or* starting points*) if it contains x_1 $start^+$ and x_2 $start^-$ and $x_1 + x_2 = x$.*

Definition 2.16 *Consider a GDH-Protocol with n participants and let $\alpha_1, \ldots, \alpha_n$ be the n histories given in the definition of this protocol. We say that $C(M_i \to M_j)$* precedes *(written $\preceq$) $C(M_i \to M_k)$ iff $\forall y : Id(\alpha_k(y)) = M_i$, $\exists x : Id(\alpha_j(x)) = M_i$ and $node(\alpha_j(x)) \preceq node(\alpha_k(y))$.*

Given a node n on s_i, we also write that $C(M_i \to M_j) \preceq n$ if $\exists x : Id(\alpha_j(x)) = M_i$ and $node(\alpha_j(x)) \preceq n$, and that $n \preceq C(M_i \to M_j)$ when $\forall x : Id(\alpha_j(x)) = M_i$, $n \preceq node(\alpha_j(x))$.

The strict precedence relation $\prec$ corresponds to the precedence relation except that we replace "$\preceq$" with "$\prec$" in its definition.

We may observe that point 2e of Def. 2.11 of GDH-Protocols implies that the precedence relation is always defined in $C(M_i \to M_j) \preceq C(M_i \to M_k)$ when $i \neq j$ and $i \neq k$.

Definition 2.17 *Consider a GDH-Protocol with n participants and $\alpha_1, \ldots \alpha_n$ the n histories given in the definition of this protocol. We define* $\text{split}(M_i, M_j)$ *as* $Id(\alpha_i(k))$ *where* $k = \max_l(\alpha_i(l) = \alpha_j(l))$ *(*$\text{split}(M_i, M_j)$ *is undefined if* $\alpha_i(l) \neq \alpha_j(l)\ \forall l$*).*

We say that the product of contributions $\prod_{i\in\mathcal{I}} C^{-1}(M_{j_i} \to M_{k_i}) \cdot C(M_{j_i} \to M_{l_i})$ *(with $\mathcal{I}$ a set of indices, $1 \leq j_i, k_i, l_i \leq n$) contains x* splits *(or* splitting points*) if there exist x indices in $\mathcal{I}$ such that* $split(M_{k_i}, M_{l_i}) = M_{j_i}$*.*

These definitions are used in the following proposition in which we state sufficient conditions for the possibility of building pairs of elements of G more complex than those described in the previous subsection.

Proposition 2.13 *Consider a GDH-Protocol with n participants and let* $p = \prod_{i\in\mathcal{I}} C^{-1}(M_{j_i} \to M_{k_i}) \cdot C(M_{j_i} \to M_{l_i})$ *(with $1 \leq j_i, k_i, l_i \leq n$) be a product of contributions such that all pairs of contributions are provided in different strands. Then an active attacker can obtain a pair (g_1, g_2) of elements of G such that $g_2 = g_1^p$ if one of the following conditions is verified:*

1. *p contains at most one splitting point and no starting point;*
2. *p contains no splitting point, one $start^+$ and no $start^-$;*
3. *p contains no splitting point, no $start^+$ and one $start^-$;*
4. *p contains no splitting point, one $start^+$ and one $start^-$; both occurring for the index $i \in \mathcal{I}$;*
5. *p contains no splitting point, one $start^+$ (for the index $i_+ \in \mathcal{I}$), one $start^-$ (for the index $i_- \in \mathcal{I}$, $i_+ \neq i_-$) and $C(M_{j_{i_-}} \to M_{k_{i_-}}) \prec C(M_{j_{i_-}} \to M_{l_{i_-}})$ or $C(M_{j_{i_+}} \to M_{l_{i_+}}) \prec C(M_{j_{i_+}} \to M_{k_{i_+}})$.*

Proof. We consider these sufficient conditions successively.

1. p contains at most one splitting point and no starting point

Let $m \in \mathcal{I}$ be the index such that $split(M_{k_m}, M_{l_m}) = M_{j_m}$, or m be a random element of $\mathcal{I}$ if p does not contain splitting point. If we initialize g_1 and g_2 to α (or to any random value) and execute Algorithm 4 for the product $C^{-1}(M_{j_m} \to M_{k_m}) \cdot C(M_{j_m} \to M_{l_m})$, we obtain a first pair (g_1, g_2).

Then, we may successively apply Algorithm 4 for all products of contributions corresponding to indexes in $i \in \mathcal{I}\backslash\{m\}$, providing the

values obtained for g_1 and g_2 at the end of each execution as input for the next one.

The correctness of this procedure relies on the fact that Algorithm 4 will transform pairs (g_1, g_2) such that $g_2 = g_1^{p_x}$ into pairs (g_1, g_2) such that $g_2 = g_1^{p_x \cdot C^{-1}(M_{j_i} \to M_{k_i}) \cdot C(M_{j_i} \to M_{l_i})}$ for each value of i, which can be verified to be correct when the considered product of contributions does not contain any splitting point or starting point.

2. p contains no splitting point, one start$^+$ and no start$^-$

The process is nearly identical to the above. Let $m \in \mathcal{I}$ be the index such that $start(M_{l_m}) = M_{j_m}$. If we initialize g_1 and g_2 to α and execute Algorithm 4 for the product $C^{-1}(M_{j_m} \to M_{k_m}) \cdot C(M_{j_m} \to M_{l_m})$, we obtain a first pair (g_1, g_2).

Then, we may successively execute Algorithm 4 for all products of contributions corresponding to indexes in $i \in \mathcal{I} \backslash \{m\}$, providing the values obtained for g_1 and g_2 at the end of each execution as input for the next one.

The correctness of this procedure relies on the same observations as above.

3. p contains no splitting point, no start$^+$ and one start$^-$

The procedure is the same as the previous one.

4. p contains no splitting point, one start$^+$ and one start$^-$; both occurring for the index $i \in \mathcal{I}$

The process suggested in the treatment of this proposition first condition may also be applied in this case.

5. p contains no splitting point, one start$^+$ (for the index $i_+ \in \mathcal{I}$), one start$^-$ (for the index $i_- \in \mathcal{I}$) and $C(M_{j_{i_-}} \to M_{k_{i_-}}) \prec C(M_{j_{i_-}} \to M_{l_{i_-}})$ or $C(M_{j_{i_+}} \to M_{l_{i_+}}) \prec C(M_{j_{i_+}} \to M_{k_{i_+}})$

Suppose $C(M_{j_{i_-}} \to M_{k_{i_-}}) \prec C(M_{j_{i_-}} \to M_{l_{i_-}})$ and $node(\alpha_{k_{i_-}}(1)) = \langle s_{j_{i_-}}, \hat{z} \rangle$. We initialize g_1 and g_2 to α and apply Algorithm 4 for the product $C^{-1}(M_{j_{i_-}} \to M_{k_{i_-}}) \cdot C(M_{j_{i_-}} \to M_{l_{i_-}})$ until $z = \hat{z}$ (we also execute this algorithm for this value of z). At this point, our precedence assumption guarantees us that $g_1 = \langle \alpha_k, 1 \rangle$ and $g_2 = \alpha$. Then, we execute the same algorithm for the product $C^{-1}(M_{j_{i_+}} \to M_{k_{i_+}}) \cdot C(M_{j_{i_+}} \to M_{l_{i_+}})$ and finally complete the first execution of the algorithm for the values of z going from $\hat{z} + 1$ to $length(s_j)$ with the updated values of g_1 and g_2. We may now execute Algorithm 4 for the indexes $i \in \mathcal{I} \backslash \{i_+, i_-\}$, always updating g_1 and g_2, which provides us the desired pair. The case $C(M_{j_{i_+}} \to M_{l_{i_+}}) \prec C(M_{j_{i_+}} \to M_{k_{i_+}})$ can be managed symmetrically. ∎

The first four conditions stated in this proposition only involve values of splitting or starting points. The following corollary gives a similar condition allowing us to manage a particular case of the fifth sufficient condition.

Corollary 2.14 *The following condition is sufficient to make conditions (4) or (5) of Proposition 2.13 correct:*

- *p contains no splitting point, one start$^+$ (for the index $i_+ \in \mathcal{I}$), one start$^-$ (for the index $i_- \in \mathcal{I}$), $k_{i_+} = k_{i_-}$ and $l_{i_+} = l_{i_-}$.*

Proof. When $i_+ = i_-$, this condition implies condition (4). We now assume $i_+ \neq i_-$, and complete the proof of our corollary by verifying that at least one of the two following relations must be true:

- $C(M_{j_{i_-}} \rightarrow M_{k_{i_-}}) \prec C(M_{j_{i_-}} \rightarrow M_{l_{i_-}})$
- $C(M_{j_{i_+}} \rightarrow M_{l_{i_+}}) \prec C(M_{j_{i_+}} \rightarrow M_{k_{i_+}})$

Let $k = k_{i_+} = k_{i_-}$, $l = l_{i_+} = l_{i_-}$, and let x and y be the smallest indexes such that $Id(\alpha_k(x)) = M_{j_{i_+}}$ and $Id(\alpha_l(y)) = M_{j_{i_-}}$ (if one of these values is not defined, then the corresponding precedence relation is trivially verified). From the definition of histories and since $C(M_{j_{i_-}} \rightarrow M_l)$ and $C(M_{j_{i_+}} \rightarrow M_k)$ do not contain starting points, we know that $node(\alpha_k(1)) \prec node(\alpha_k(x))$ and that $node(\alpha_l(1)) \prec node(\alpha_l(y))$. Assume now that $C(M_{j_{i_-}} \rightarrow M_k)$ does not strictly precede $C(M_{j_{i_-}} \rightarrow M_l)$, which implies that $node(\alpha_l(y)) \preceq node(\alpha_k(1))$. If this is true, the first two precedence relations we stated directly implies that $node(\alpha_l(1)) \prec node(\alpha_k(x))$ and therefore that $C(M_{j_{i_+}} \rightarrow M_l) \prec C(M_{j_{i_+}} \rightarrow M_k)$. This proves the assertion at the beginning of this proof, which implies that the fifth condition of Proposition 2.13 is satisfied. ∎

We now have six sufficient conditions for the obtention of pairs of the form required for the application of Theorem 2.10. These conditions are not necessary at all: we can imagine an history starting with the sending of the public value α (as it is the case in the A-GDH.2 protocol [7] for instance), and this starting point clearly does not need to be taken into account in the last four sufficient conditions of Proposition 2.13.

We now exploit these sufficient conditions to see whether it is possible to build a pair of the form given in Theorem 2.10.

5.3. Obtaining a Secret Pair

The following theorem shows that, for any GDH-Protocol with at least four participants, it is possible to obtain a pair of the form given in Theorem 2.10.

Theorem 2.15 *For any GDH-Protocol with at least four participants, it is possible for an active attacker to obtain a pair (g_1, g_2) of elements of G such that $g_2 = g_1^p$ where*

$$\begin{aligned} p \;=\; & C^{-1}(M_i \to M_i) \cdot C(M_i \to M_j) \cdot \\ & [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \to M_i) \cdot C(M_l \to M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \to M_i) \cdot C(M_l \to M_j)] \cdot \prod_{l \in 1 \ldots n} K_{Il}^{e_l} \end{aligned}$$

for some choice of M_i, M_j, M_k, S_j, S_k and e_l; where M_i, M_j and M_k are three different members of the group M while S_j and S_k are two disjoint sets of users such that $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j$, $M_i \notin \mathsf{S}_k$ and $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$.

Proof. We prove that the product p above respects the conditions stated in Proposition 2.13 for a certain choice of M_i, M_j, M_k, S_j and S_k. We first try to proceed by exhaustive search. Given a GDH-Protocol, we randomly choose four histories, say α_1, α_2, α_3 and α_4. We can represent these histories as forests (i.e. as sets of trees): the roots represent the starting points of histories, the internal nodes their splitting points and the leafs the end of the histories. It can be verified that there are only six different ways to build a binary forest (i.e. a forest of binary trees) with four leafs: we represented these forests in Fig. 2.7. For each of the six cases, the roots of the trees are on the left side while the leafs are on the right side. We labelled the leafs M_1, M_2, M_3, M_4; the leaf labelled M_i corresponding to the last node of α_i, i.e. $node(\alpha_i^F)$. A tree having a leaf labelled M_i has a root labelled M_j iff $start(M_i) = M_j$. Finally, if there exist a node that is the root of the smallest subtree containing the leafs labelled M_i and M_j, then this node is labelled with $split(M_i, M_j)$.

This representation does not directly allow to represent all configurations of histories we may encounter:

- We only consider binary trees, as if the histories were only split into two parts during their execution. It is however possible that a user M_x receiving an element of $\mathcal{G}$ as $\langle \alpha_i, s \rangle = \langle \alpha_j, s \rangle = \langle \alpha_k, s \rangle$ during an execution of the protocol exponentiates it with three different values that are sent as $\langle \alpha_i, s+1 \rangle \neq \langle \alpha_j, s+1 \rangle \neq \langle \alpha_k, s+1 \rangle$. This situation would have been naturally represented through the tree in Fig.2.8 (a). However, we are only interested in the *start* and *split* properties of histories and it can be easily verified that replacing the part of tree in Fig.2.8 (a) with the binary tree in Fig.2.8 (b) does not change these

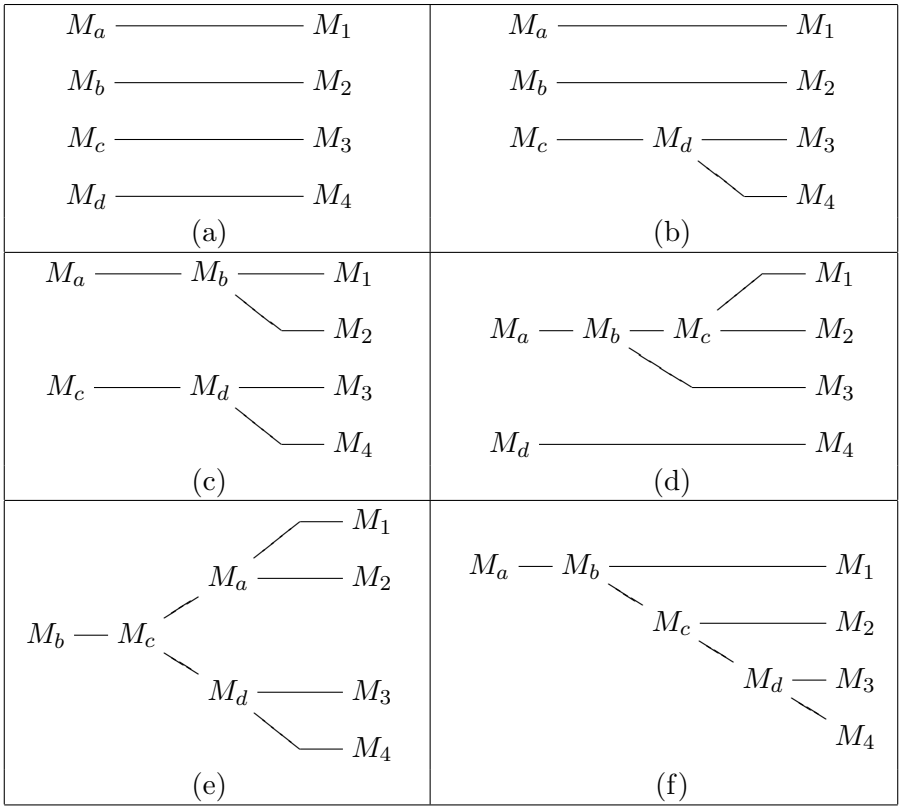


FIGURE 2.7. Six varieties of binary forests with four leafs.

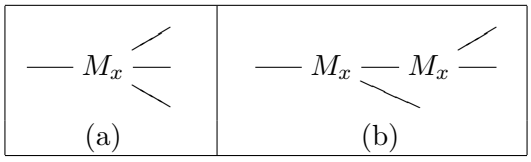


FIGURE 2.8. Simplification of non-binary trees.

properties. This can trivially be transposed for nodes where histories are split into more than three parts.

- A splitting point may be the end of an history, giving a forest with three leafs (or less) instead of four. The corresponding trees may however be seen as trees represented in Fig. 2.7 where some branches have zero length.

So, it is always possible to represent the *split* and *start* properties of four chosen histories through one of the six forest of Fig.2.7.

We may observe that, for each of these forests, there are exactly four nodes that are splitting or starting points (they are labelled M_a, M_b, M_c

and M_d). This can be justified easily: each history has a starting point, and when two histories are sharing the same starting point, then there must exist a splitting point for these histories since they cannot have the same end ($R_i \neq R_j$ when $i \neq j$). Each of these four nodes may be labelled with the identifier of any group member, and our objective will be to prove that for any of the six forests and for any value of these labels, it is possible to choose M_i, M_j, M_k, S_j and S_k such that at least one of the conditions expressed in Proposition 2.13 or Corollary 2.14 is verified.

We check the first three conditions of Proposition 2.13 and the one of Corollary 2.14 by exhaustive search, considering that each of the *start* and *split* nodes may have five different values: M_1, M_2, M_3, M_4, M_x; this last value representing a node labelled with an identifier different of M_1, M_2, M_3 and M_4. We also considered two possible choices of S_j and S_k: $\mathsf{S}_j = \mathsf{M}\backslash\{M_i, M_j\}$ and $\mathsf{S}_k = \{M_j\}$ or $\mathsf{S}_j = \{M_k\}$ and $\mathsf{S}_k = \mathsf{M}\backslash\{M_i, M_k\}$. The consideration of a single label M_x for all values different of M_1, M_2, M_3 and M_4 is not a restriction since, for the two considered choices of S_j and S_k, the product of contributions of users represented by M_x is always the same ($C^{-1}(M_x \rightarrow M_i) \cdot C(M_x \rightarrow M_j)$ or $C^{-1}(M_x \rightarrow M_i) \cdot C(M_x \rightarrow M_k)$) according to the way S_j and S_k are defined.

This verification is illustrated in Algorithm 5, which uses the function *CheckFourConditions* described in Algorithm 6.

We executed this algorithm and verified that at least one of the four considered sufficient conditions was verified for an appropriate choice of M_i, M_j, M_k, S_j and S_k excepted in nine cases, all of them in the forest represented in Fig.2.7(a). The corresponding values of M_a, M_b, M_c and M_d are represented in Table 2.1.

TABLE 2.1. Problematic values of M_a, M_b, M_c and M_d

	M_a	M_b	M_c	M_d
1)	M_2	M_1	M_4	M_3
2)	M_2	M_3	M_4	M_1
3)	M_2	M_4	M_1	M_3
4)	M_3	M_1	M_4	M_2
5)	M_3	M_4	M_1	M_2
6)	M_3	M_4	M_2	M_1
7)	M_4	M_1	M_2	M_3
8)	M_4	M_3	M_1	M_2
9)	M_4	M_3	M_2	M_1

So, since $\prod_{l \in 1 \ldots n} K_{ll}^{e_l}$ is a product of keys the attacker knows, our theorem is proved for all cases except these nine.

Algorithm 5 Returns a list of the forests and values of M_a, M_b, M_c and M_d for which a choice of M_i, M_j, M_k, S_j and S_k making the product p defined in the wording of Thm. 2.15 to respect one of the conditions of Proposition 2.13 has not been found.

for all *Forest* **in** Fig. 2.7 **do**
  for all M_a, M_b, M_c, M_d, each chosen **in** $\{M_1, M_2, M_3, M_4, M_x\}$ **do**
    $Solution$:= False
    for $i, j, k := 1$ to 4 **do**
      if $M_i \neq M_j$ **and** $M_i \neq M_k$ **and** $M_j \neq M_k$ **then**
        $\mathsf{S}_j := \{M_1, M_2, M_3, M_4, M_x\} \backslash \{M_i, M_j\}$ $\mathsf{S}_k := \{M_j\}$
        if $CheckFourConditions()$ **then**
          $Solution$:= True
          $i, j, k := 4$
        end if
        $\mathsf{S}_j := \{M_k\}$ $\mathsf{S}_k := \{M_1, M_2, M_3, M_4, M_x\} \backslash \{M_i, M_k\}$
        if $CheckFourConditions()$ **then**
          $Solution$:= True
          $i, j, k := 4$
        end if
      end if
    end for
    if $Solution$ = False **then**
      Write "The current forest is problematic for the current choice of M_a, M_b, M_c and M_d"
    end if
  end for
end for

We now verify that it is possible to choose M_i, M_j, M_k, S_j and S_k for the protocols the structure of which corresponds to the ninth description of Table 2.1 in such a way that the fifth condition of Proposition 2.13 is satisfied. The other cases will be discussed further.

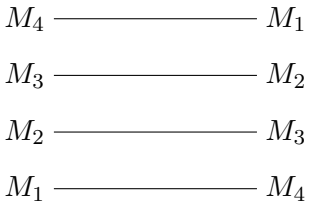


FIGURE 2.9. A problematic forest

We are in the presence of the forest represented in Fig. 2.9.

Algorithm 6 Given a forest, M_a, M_b, M_c, M_d, M_i, M_j, M_k, S_j and S_k, returns "True" if the product p defined in the wording of Thm. 2.15 respects one of the first three conditions of Prop. 2.13 or the condition of Corollary 2.14.

function $b := CheckFourConditions()$
for the current forest and values of M_a, M_b, M_c, M_d, M_i, M_j, M_k, S_j, S_k, **do**
$p := C^{-1}(M_i \rightarrow M_i) \cdot C(M_i \rightarrow M_j) \cdot$
$[\mathsf{S}_j \backslash M_I : C^{-1}(M_i \rightarrow M_j) \cdot C(M_i \rightarrow M_k)] \cdot$
$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_k)] \cdot$
$\prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_j)]$
if p contains at most one splitting point and no starting point **then**
Return True
else if p contains no splitting point, one start$^+$ and no start$^-$ **then**
Return True
else if p contains no splitting point, no start$^+$ and one start$^-$ **then**
Return True
else if p contains no splitting point, one start$^+$ and one start$^-$, these two starting points occurring in pairs of contributions intended to the same two users **then**
Return True
end if
Return False
end function

Suppose we choose $M_i = M_1$, $M_j = M_2$, $M_k = M_4$, $\mathsf{S}_j = \{M_4\}$ and $\mathsf{S}_k = \mathsf{M} \backslash \{M_1, M_2\}$. This choice implies that the product p contains one start$^+$ (i.e. $C(M_1 \rightarrow M_4)$), one start$^-$ (i.e. $C(M_4 \rightarrow M_1)$), and no splitting point. If this choice satisfies the fifth condition of Proposition 2.13, the attacker is able to obtain the desired pair. If this condition is not verified, we know that $C(M_1 \rightarrow M_2) \preceq C(M_1 \rightarrow M_4)$ and that $C(M_4 \rightarrow M_2) \preceq C(M_4 \rightarrow M_1)$. Furthermore, from the definition of possible histories and from the fact that $C(M_4 \rightarrow M_1)$ is a starting point, we can write:

$$node(\alpha_2(1)) \prec C(M_4 \rightarrow M_2) \preceq node(\alpha_1(1))$$

Suppose now we choose $M_i = M_2$, $M_j = M_1$, $M_k = M_3$, $\mathsf{S}_j = \{M_3\}$ and $\mathsf{S}_k = \mathsf{M} \backslash \{M_1, M_2\}$. This choice implies that the product p contains one start$^+$ (i.e. $C(M_2 \rightarrow M_3)$), one start$^-$ (i.e. $C(M_3 \rightarrow M_2)$), and no splitting point. If this choice does not satisfy the fifth condition of Proposition 2.13, $C(M_2 \rightarrow M_1) \preceq C(M_2 \rightarrow M_3)$ and $C(M_3 \rightarrow M_1) \preceq C(M_3 \rightarrow M_2)$. Furthermore, from the definition of possible histories

and from the fact that $C(M_3 \to M_2)$ is a starting point, we can write:

$$node(\alpha_1(1)) \prec C(M_3 \to M_1) \preceq node(\alpha_2(1))$$

which is in contradiction with the relation $node(\alpha_2(1)) \prec node(\alpha_1(1))$ obtained above.

Therefore, one of the two choices of M_i, M_j, M_k, S_j and S_k we proposed must verify the fifth condition of Proposition 2.13.

A similar reasoning can be carried out for the eight remaining problematic cases, and we give adequate choices for the five variables in Table 2.2.

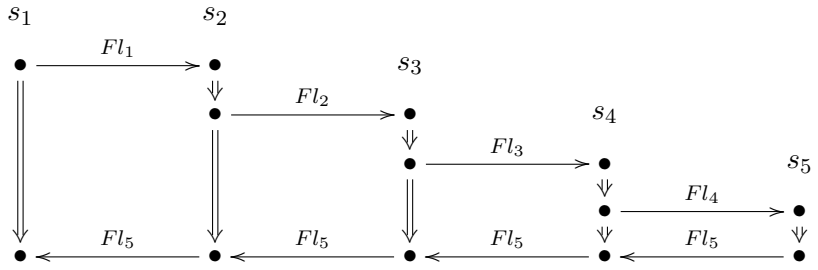
TABLE 2.2. Possible choices for M_i, M_j, M_k, S_j and S_k

	M_a	M_b	M_c	M_d	M_i	M_j	M_k	S_j	S_k
1)	M_2	M_1	M_4	M_3	M_1	M_3	M_2	$\{M_2\}$	$\mathsf{M}\backslash\{M_1, M_2\}$
					M_3	M_1	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_3, M_4\}$
2)	M_2	M_3	M_4	M_1	M_3	M_1	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_3, M_4\}$
					M_3	M_4	M_1	$\mathsf{M}\backslash\{M_3, M_4\}$	$\{M_4\}$
3)	M_2	M_4	M_1	M_3	M_1	M_2	M_4	$\mathsf{M}\backslash\{M_1, M_2\}$	$\{M_2\}$
					M_1	M_4	M_2	$\{M_2\}$	$\mathsf{M}\backslash\{M_1, M_2\}$
4)	M_3	M_1	M_4	M_2	M_1	M_4	M_3	$\{M_3\}$	$\mathsf{M}\backslash\{M_1, M_3\}$
					M_1	M_3	M_4	$\mathsf{M}\backslash\{M_1, M_3\}$	$\{M_3\}$
5)	M_3	M_4	M_1	M_2	M_1	M_2	M_3	$\{M_3\}$	$\mathsf{M}\backslash\{M_1, M_3\}$
					M_2	M_1	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_2, M_4\}$
6)	M_3	M_4	M_2	M_1	M_1	M_2	M_3	$\{M_3\}$	$\mathsf{M}\backslash\{M_1, M_3\}$
					M_1	M_3	M_2	$\mathsf{M}\backslash\{M_1, M_3\}$	$\{M_3\}$
7)	M_4	M_1	M_2	M_3	M_1	M_3	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_1, M_4\}$
					M_1	M_4	M_3	$\mathsf{M}\backslash\{M_1, M_4\}$	$\{M_4\}$
8)	M_4	M_3	M_1	M_2	M_1	M_2	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_1, M_4\}$
					M_1	M_4	M_2	$\mathsf{M}\backslash\{M_1, M_4\}$	$\{M_4\}$
9)	M_4	M_3	M_2	M_1	M_1	M_2	M_4	$\{M_4\}$	$\mathsf{M}\backslash\{M_1, M_4\}$
					M_2	M_1	M_3	$\{M_3\}$	$\mathsf{M}\backslash\{M_2, M_3\}$

Finally, we proved that, for any GDH-protocol, it is possible to select M_i, M_j, M_k, S_j and S_k in such a way that one of the five sufficient conditions of Proposition 2.13 is satisfied for the product of contributions p. ∎

The main constructions exploited in the proof above are exemplified below.

Example 2.18 We consider the A-GDH.2 protocol executed by five parties. For the record, the bundle in Fig. 2.10 describes a session of this protocol; the histories α_1, α_2, α_3, α_4, and α_5 may easily be reconstructed from this bundle.



$$
\begin{aligned}
Fl_1 &= \alpha, \alpha^{r_1} \\
Fl_2 &= \alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1 r_2} \\
Fl_3 &= \alpha^{r_2 r_3}, \alpha^{r_1 r_3}, \alpha^{r_1 r_2}, \alpha^{r_1 r_2 r_3} \\
Fl_4 &= \alpha^{r_2 r_3 r_4}, \alpha^{r_1 r_3 r_4}, \alpha^{r_1 r_2 r_4}, \alpha^{r_1 r_2 r_3}, \alpha^{r_1 r_2 r_3 r_4} \\
Fl_5 &= \alpha^{r_2 r_3 r_4 r_5 K_{15}}, \alpha^{r_1 r_3 r_4 r_5 K_{25}}, \alpha^{r_1 r_2 r_4 r_5 K_{35}}, \alpha^{r_1 r_2 r_3 r_5 K_{45}}
\end{aligned}
$$

FIGURE 2.10. A five parties run of the A-GDH.2 protocol

As proposed in the proof of Theorem 2.15, we now consider four of the five histories, say α_1, α_2, α_3 and α_4. It can be observed that

$$start(M_1) = start(M_2) = start(M_3) = start(M_4) = M_1;$$

$$split(M_1, M_2),\ split(M_1, M_3),\ split(M_1, M_4) \text{ are not defined;}$$

$$split(M_2, M_3) = split(M_2, M_4) = M_2;\ split(M_3, M_4) = M_3.$$

This scheme corresponds to the forest represented in Fig. 2.11 which can be mapped to Fig. 2.7 (d).

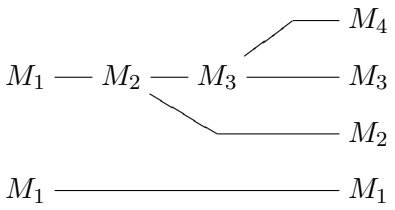


FIGURE 2.11. Forest corresponding to the first four histories in the A-GDH.2 protocol

If we look at the solution Algorithm 5 should find for that configuration, we obtain $i = 4, j = 3, k = 2$ while $\mathsf{S}_j = \mathsf{M} \backslash \{M_i, M_j\} = \{M_1, M_2, M_5\}$ and $\mathsf{S}_k = \{M_j\} = \{M_3\}$. So, it is possible to obtain a pair (g_1, g_2) of elements of G such that $g_2 = g_1^p$ where

$$
\begin{aligned}
p \;=\;& C^{-1}(M_4 \rightarrow M_4) \cdot C(M_4 \rightarrow M_3) \cdot \\
& [\{M_1, M_2, M_5\} \backslash M_I : C^{-1}(M_4 \rightarrow M_3) \cdot C(M_4 \rightarrow M_2)] \cdot \\
& [\{M_1, M_2, M_5\} \backslash M_I : C^{-1}(M_3 \rightarrow M_4) \cdot (M_3 \rightarrow M_2)] \cdot \\
& [M_3 \backslash M_I : C^{-1}(M_1 \rightarrow M_4) \cdot C(M_1 \rightarrow M_3)] \cdot \\
& [M_3 \backslash M_I : C^{-1}(M_2 \rightarrow M_4) \cdot C(M_2 \rightarrow M_3)] \cdot \\
& [M_3 \backslash M_I : C^{-1}(M_5 \rightarrow M_4) \cdot C(M_5 \rightarrow M_3)] \cdot \prod_{l \in 1 \ldots n} K_{Il}^{e_l}
\end{aligned}
$$

It can be easily checked that this product does not contain any splitting point nor any starting point, so the conditions expressed in Proposition 2.13 are verified.

We have to examine one last question before being able to claim that it is always possible to break the implicit key authentication for any GDH-Protocol: we still do not know whether an intruder that is able to obtain a pair (g_1, g_2) such that $g_2 = g_1^p$ where $p = R_i \cdot K_i$ for some M_i is also able to submit g_1 as the value M_i will use to compute his view of the group key. This question corresponds to the verification of the last two conditions expressed in Section 4.4.4 of Chapter 1:

(1) The intruder cannot interact with any nodes n such that $n = n_i^F$ or $n_i^F \Rightarrow^+ n$ for constructing g_1
(2) The intruder cannot affect any value to α_i^F when constructing g_2

We verify the first of these conditions through the theorem below.

Theorem 2.16 *Consider a GDH-Protocol with n participants. Suppose* $p = C^{-1}(M_i \rightarrow M_i) \cdot C(M_i \rightarrow M_j)$ *(with* $1 \leq i \neq j \leq n$*) for a particular session* S *of the protocol and* p' *a product of contributions to other sessions of the same protocol. If an active attacker is able to obtain a pair* (g_1, g_2) *of elements of* G *such that* $g_2 = g_1^{p \cdot p'}$*, then he is able to obtain* g_1 *without interacting with any node* n *such that* $n_i^F = n$ *or* $n_i^F \Rightarrow^+ n$*.*

Proof. p' is a product of values exchanged during sessions S_x independent of S, so the nodes exploited to obtain the values in p' are not on the same strand as n_i^F.

On the other side, p is a product of values exchanged during the session S. From point 2a of Definition 2.11 and from Definition 2.9, we know that n_i^F belongs to s_i and that $term(n_i^F) = -t$. So, we may check in Algorithm 4 that all nodes of s_i that will be exploited for constructing g_1 strictly precede n_i^F and the required condition is therefore verified. ∎

The second condition asserts that we may not use any service whose input is α_i^F when constructing g_2. If we look at the expression of $R_i \cdot K_i$

as a product of contributions and keys given at the end of Theorem 2.10, we observe that the only contribution the constitution of which could exploit α_i^F in the session we are trying to attack is $C(M_i \rightarrow M_j)$. We may also observe that if α_i^F has to be affected when collecting $C(M_i \rightarrow M_j)$, then the last element of α_i is also part of α_j and, therefore, $split(M_i, M_j) = M_i$. For that reason, we will verify our second (and last) condition by proving that the Theorem 2.15 remains correct if we add a supplementary condition on the choice of M_i, M_j and M_k: we require that $split(M_i, M_j) \neq M_i$.

Theorem 2.17 *For any GDH-Protocol with at least four group members, it is always possible for an active attacker to obtain a pair (g_1, g_2) of elements of G such that $g_2 = g_1^p$ where*

$$\begin{aligned} p \;=\; & C^{-1}(M_i \rightarrow M_i) \cdot C(M_i \rightarrow M_j) \cdot \\ & [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \rightarrow M_j) \cdot C(M_i \rightarrow M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_j)] \cdot \prod_{l \in 1 \ldots n} K_{Il}^{e_l} \end{aligned}$$

for some choice of M_i, M_j, M_k, S_j, S_k and e_l; where M_i, M_j and M_k are three different members of the group M; S_j and S_k are two disjoint sets of users such that $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j$, $M_i \notin \mathsf{S}_k$ and $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$; and $split(M_i, M_j) \neq M_i$.

Proof. The proof of this theorem is the same as the one of Theorem 2.15 except that we have one more condition to verify in the choice of M_i and M_j: $split(M_i, M_j) \neq M_i$.

This can be very easily performed by adding a **if** control structure in Algorithm 5, which becomes Algorithm 7.

When we executed this algorithm, we anew obtained the same nine problematic cases, which are not concerned with the present restriction (since $split(M_i, M_j)$ is undefined in the corresponding forest).

∎

6. Concluding Remarks

6.1. Summary

In this chapter, we analyzed a family of authenticated group key agreement protocols, family that we defined as a generalization of the GDH protocols proposed in the context of the Cliques project.

The main result of the chapter is the proof that it is impossible to write a protocol of this family providing implicit key authentication as

Algorithm 7 Returns a list of the forests and values of M_a, M_b, M_c and M_d for which a choice of M_i, M_j, M_k, S_j and S_k making the product p defined in the wording of Thm. 2.15 to respect one of the conditions of Proposition 2.13 has not been found.

for all *Forest* **in** Fig. 2.7
for all M_a, M_b, M_c, M_d, each chosen **in** $\{M_1, M_2, M_3, M_4, M_x\}$ **do**
  $Solution$:= False
  for $i, j, k := 1$ to 4 **do**
    if $M_i \neq M_j$ **and** $M_i \neq M_k$ **and** $M_j \neq M_k$ **then**
      if $split(M_i, M_j) \neq M_i$ **then**
        $\mathsf{S}_j := \{M_1, M_2, M_3, M_4, M_x\} \backslash \{M_i, M_j\}$ $\mathsf{S}_k := \{M_j\}$
        if $CheckFourConditions()$ **then**
          $Solution$:= True
          $i, j, k := 4$
        end if
        $\mathsf{S}_j := \{M_k\}$ $\mathsf{S}_k := \{M_1, M_2, M_3, M_4, M_x\} \backslash \{M_i, M_k\}$
        if $CheckFourConditions()$ **then**
          $Solution$:= True
          $i, j, k := 4$
        end if
      end if
    end if
  end for
  if $Solution$ = False **then**
    Write "The current forest is problematic for the current choice of M_a, M_b, M_c and M_d"
  end if
end for
end for
Return True

soon as it is executed by at least four participants. This proof being established all along the chapter, we gather the main points here.

We prove our result by providing a systematic way to set up a scenario that undermines the IKA property. The process is as follows.

Consider a GDH-Protocol executed by a group M of n users such that $n \geq 4$ and $M_I \notin \mathsf{M}$. The intruder selects:

- three members of M: M_i, M_j and M_k
- two disjoint sets of users S_j and S_k such that $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j \cup \mathsf{S}_k$, $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$.

This selection must also respect the two following conditions:

- the product $p =$

  $$C^{-1}(M_i \rightarrow M_i) \cdot C(M_i \rightarrow M_j) \cdot$$
  $$[\mathsf{S}_j \backslash M_I : C^{-1}(M_i \rightarrow M_j) \cdot C(M_i \rightarrow M_k)] \cdot$$
  $$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot (M_l \rightarrow M_k)] \cdot$$
  $$\prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \rightarrow M_i) \cdot C(M_l \rightarrow M_j)] \cdot \prod_{M_l \in \mathsf{M}} K_{Il}^{e_l}$$

  respects at least one of the conditions described in Proposition 2.13.
- $split(M_i, M_j) \neq M_i$

Theorem 2.17 guarantees that the choice of such M_i, M_j, M_k, S_j and S_k is always possible, and a systematic algorithm allowing this selection is provided in its proof (we guess that an heuristic choice is however more convenient in the practice).

After having selected these values, the intruder may build a pair (g_1, g_2) such that $g_2 = g_1^p$ by applying the algorithms given in the proof of Proposition 2.13, and replace the value M_i will use to compute the group key (i.e. α_i^F) with g_1.

At this time, and given that $p = R_i \cdot K_i$ as we proved in Theorem 2.10, M_i will compute g_2 as his view of the group key, which is in contradiction with the implicit key authentication property.

6.2. Discussion of the Results

6.2.1. Cardinality of the group

A first interesting point is that our result is only valid for protocols executed by at least four users. This shows that the attacks we discovered are really attacks against group protocols and emphasizes the need to consider these protocols differently than simple extensions of two-party ones.

If we examine the two-party version of the A-GDH.2 protocol (represented in Fig. 2.12), we are not able to undermine the IKA property.

$$M_1 \underset{\alpha^{r_2 K_{12}}}{\overset{\alpha^{r_1}}{\rightleftarrows}} M_2 \qquad K = \alpha^{r_1 r_2}$$

FIGURE 2.12. A-GDH.2 Key Agreement protocol

Informally, if we go back to the consideration of the previous chapter, we can justify the security of this protocol as follows.

The set of secret ratios P_S is $\{r_1K_{12}^{-1}, r_2\}$. We now try to write $r_1K_{12}^{-1}$ as a combination of services. r_1 is only provided in the session we want to attack, so we have to use the r_1 service provided by M_1 in that session. We now have to obtain the K_{12}^{-1} part of the secret $r_1K_{12}^{-1}$. K_{12} is part of two kinds of services: the reply of M_1 in sessions executed by M_2 and M_1, and the reply of M_2 in sessions executed by M_1 and M_2. In both cases, K_{12} is provided together with a random value uniquely originating: the services containing K_{12} have the form $r_1'K_{12}$ or $r_2'K_{12}$, and they are the only ones containing these random values. So, we are not able to obtain the secret $r_1K_{12}^{-1}$. A similar reasoning may be carried out about the secret r_2.

So, we have a two-party protocol for which we are not able to write the secret ratios as combination of services. We now consider a three-party protocol which also presents the implicit key authentication property (in our model): the Tri-GDH protocol. For the record, a run of this protocol is represented in Fig. 2.13. The three central messages are exchanged first, while the three external are computed from those just received.

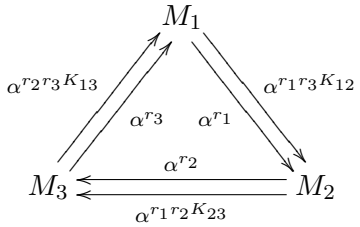


FIGURE 2.13. A run of the Tri-GDH protocol

For this protocol, the set of secret ratios P_S is defined as $\{r_1K_{13}^{-1}$, $r_2K_{12}^{-1}$, $r_3K_{23}^{-1}\}$. Theorem 2.10 asserts that it is possible to write these secrets as a combination of services. We now informally show why these services cannot be used to obtain an attack against this protocol.

Consider the secret ratio $r_1K_{13}^{-1}$. The K_{13}^{-1} part can only be obtained through some service r_xK_{13} used in the negative direction. The r_x^{-1} value can only be compensated by the service r_x of the same session used in the positive direction.

At this point, we have obtained the K_{13}^{-1} part of the secret. We now turn to the r_1 part.

The service r_1 could be used in the positive direction, but it is a starting service and we already used the starting service r_x in the positive direction, so this choice does not allow to build an attack.

We could also use the $r_1 K_{12}$ service, but K_{12} can only be compensated through a service $r_y K_{12}$ used in the negative direction, and the only other service providing r_y is r_y that is also a starting service; so we also cannot use it in the positive direction.

This protocol being completely symmetric, these arguments may be transposed to the other secret ratios of this protocol.

So, in the case of the Tri-GDH protocol, the security relies on the conditions for the composition of services rather than on the impossibility of writing the secret ratios as a combination of them.

These considerations intuitively justify why the result of this chapter is only valid for groups with cardinality greater than four.

6.2.2. Verification of the seven conditions of Chapter 1

In Section 4.4.4 of Chapter 1, we described seven conditions on the way services have to be offered in order to be usable to attack a protocol.

These conditions were stated in a more restrictive context than the one of this chapter: we considered that the messages sent by a user executing a protocol were directly built from values they just received. In the context of GDH-Protocols however, a user can use any message he received to build new terms. The seven conditions of Chapter 1 must nevertheless be respected when building our attacks against GDH-Protocols, as we show below.

The first condition requires that at most two values can be exploited on a single node. This condition is verified in our attacks since we collect at most two contributions on each strand.

The second condition imposes restrictions on the number of splitting and starting points we can exploit. It is transposed (on a stronger form) in the sufficient conditions of Proposition 2.13.

The third condition states that we cannot exploit values exchanged on nodes occurring before a splitting point. As the first one, this condition is verified because we consider at most two contributions per strand.

The fourth condition states that, when exploiting a starting point to obtain an element of G, we cannot use any value exchanged on the same strand and before this point to construct this element. This condition is verified because the two contributions we are using on a strand are collected in opposite direction and because all nodes on an history occur after the starting point of this history.

The fifth condition corresponds to the restriction we stated in the fourth point of Proposition 2.13 about the use of two starting points.

Finally, the last two conditions are explicitly verified in Theorems 2.16 and 2.17 at the end of the previous section.

6.2.3. Number of sessions to be considered

The wording of Proposition 2.13 requires that all pairs of contributions of the product

$$\begin{aligned} p \;=\; & C^{-1}(M_i \to M_i) \cdot C(M_i \to M_j) \cdot \\ & [\mathsf{S}_j \backslash M_I : C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \to M_i) \cdot (M_l \to M_k)] \cdot \\ & \prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \to M_i) \cdot C(M_l \to M_j)] \cdot \prod_{M_l \in \mathsf{M}} K_{Il}^{e_l} \end{aligned}$$

must belong to different strands.

We may therefore observe that three sessions of the protocol are sufficient to execute our attacks:

(1) a first necessary session is the session executed by the group M, in order to collect the services corresponding to the contributions of the first line.
(2) a second group is formed by all members of M except those of S_j which are replaced by M_I, and we need to collect the contributions of the two central lines of the expression of p for this group. Since $M_i \notin \mathsf{S}_j$, all pairs of contributions of these two lines are provided by different group members, and one session of the protocol is sufficient to collect them.
(3) a last session is executed by all members of M except those of S_k which are replaced by M_I, and we need to collect the contributions of the last line of the expression of p for this group. Anew, all pairs of contribution of this line are provided by different group members, and one session of the protocol is sufficient to collect them.

Three sessions of a protocol are then required to perform the attacks described in this chapter. This is however a minimum value (in the worst case), and up to $n + 1$ sessions may be considered since p is a product of $n + 1$ pairs of contributions.

6.3. Conclusion

The results of this chapter emphasize once more the interest of being able to systematically reason about security protocols: our model allowed us to obtain general results about a family of protocols including several published ones. Within the scope of logical approaches, the most closely related results are probably those presented in [24, 25, 28, 29] about the security of ping-pong protocols. As far as we know, similar works exploiting computational approaches do not exist.

These developments were carried out in the specific case of the GDH-Protocols. It would be interesting to examine in which measure specific aspects of our approach could be adapted to other protocol families. We think for instance to the separation of the algebraic problem of rewriting secrets from the combinatorial problem raised by the message routing constraints.

CHAPTER 3

Design and Analysis of an AGKAP

1. Introduction

In this thesis first chapter, we analyzed several authenticated group key agreement protocols and discovered flaws in each of them. In the second chapter, we tried to fix these protocols, and showed it was in fact not possible, at least without modifying the proposed design rules. We now design and analyze a new authenticated group key agreement protocol: the AT-GDH protocol.

This protocol is based on the classical key construction through a ring structure, originally presented in 1996 by Steiner, Tsudik and Waidner for the GDH.2 protocol [73]. However, contrary to what was proposed for the (S)A-GDH.2 protocols [6, 7], the authentication services are obtained through the use of a signature scheme, and the freshness of the keying material is insured through the use of a nonce and a hash function. Furthermore, achieving the different security goals through these cryptographic primitives will make our analysis more convenient, at the cost of relying on a greater number of intractability assumptions.

2. Basic Scheme and Security Requirements

Our protocol finds its roots in the GDH.2 scheme proposed in [73], and, as for this protocol, the confidentiality of the constructed group key relies on the hardness of the Group Decisional Diffie-Hellman problem, itself implied by the hardness of the classical Decisional Diffie-Hellman problem [16, 73].

As in the previous chapters, we assume α to be a public generator of a cyclic group $\mathcal{G}$ of prime order q and r_i to be a random value selected by M_i in $\mathbb{Z}_q^*$. The GDH.2 scheme is executed as follows for a group M of 3 members M_1, M_2 and M_3:

$$\begin{aligned} M_1 \rightarrow M_2 &: \alpha, \alpha^{r_1} \\ M_2 \rightarrow M_3 &: \alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1 r_2} \\ M_3 \rightarrow M_1, M_2 &: \alpha^{r_2 r_3}, \alpha^{r_1 r_3} \end{aligned}$$

At the end of this protocol, each group member M_i is able to compute the group key $\alpha^{r_1 r_2 r_3}$ by exponentiating an element of $\mathcal{G}$ he received with his own contribution r_i.

We concentrate our analysis on three particular security goals discussed in Chapter 1:

Implicit Key Authentication: when he completed his role in a session of the protocol, each $M_i \in \mathsf{M}$ is assured that no party $M_I \notin \mathsf{M}$ can learn the key $S_n(M_i)$ (i.e. M_i's view of the key) unless helped by a dishonest member of M.

Resistance to Known Session-Secret Attacks: the compromise of past session-secrets does not allow impersonation by an active adversary in the future.

Individual Forward Secrecy: the compromise of long-term keys does not compromise past session keys, assuming that some group members may have been subject to attacks in the past, leaving the other group members unaffected.

The implicit key authentication will be insured by showing that the group key computed by any group member M_i is equal to $\alpha^{r_1 r_2 r_3}$, where r_1, r_2 and r_3 were generated by M_1, M_2 and M_3, and that $\alpha^{r_1 r_2 r_3}$ cannot be exposed on the network.

The resistance to known session-secret attacks will be insured by showing that all random contributions used to compute the group key are fresh (and therefore cannot have been compromised).

The individual forward secrecy will be trivially achieved, since we will not encrypt any message: the compromise of long-term secret will therefore not disclose any new information about past sessions.

3. Theoretical Background

Our analysis is carried out by exploiting the strand space paradigm and the authentication tests. We now introduce the corresponding theory; a more detailed description can be found in [32, 77]. Most of the text below is taken from [33].

3.1. Strand Spaces and Bundles

Consider a set A, the elements of which, called terms, are the possible messages to be exchanged between principals in a protocol.

The set of terms A is assumed to be freely generated from two disjoint sets:

- $\mathsf{T} \subseteq \mathsf{A}$ which contains texts (i.e. atomic messages);

- $\mathsf{K} \subseteq \mathsf{A}$ which contains keys. This set is equipped with a unary operator $\mathbf{inv} : \mathsf{K} \to \mathsf{K}$. We assume that **inv** is an inverse mapping the signing key to the verification key (and conversely) in a signature scheme.

Compound terms are built by two operations:

- $\mathbf{sig} : \mathsf{K} \times \mathsf{A} \to \mathsf{A}$
- $\mathbf{join} : \mathsf{A} \times \mathsf{A} \to \mathsf{A}$

We will write $\mathbf{inv}(K)$ as K^{-1}, $\mathbf{sig}(K, m)$ as $\{\!|m|\!\}_K$ and $\mathbf{join}(a, b)$ as a, b.

A *subterm* relation $\sqsubset$ is defined on A:

Definition 3.1 *If $a \in \mathsf{A}$, the subterm relation $\sqsubset$ is defined inductively, as the smallest relation such that:*

- $a \sqsubset a$;
- $a \sqsubset \{\!|g|\!\}_K$ *if* $a \sqsubset g$;
- $a \sqsubset g, h$ *if* $a \sqsubset g$ *or* $a \sqsubset h$.

By this definition, for $K \in \mathsf{K}$, we have $K \sqsubset \{\!|g|\!\}_K$ only if $K \sqsubset g$ already.

Given these definitions of A and $\sqsubset$, we can use the strands and bundles definitions 2.4, 2.5, 2.6, 2.7, 2.8 of the previous chapter. We complete these definitions as follows:

Definition 3.2 *Fix a strand space Σ the nodes of which form the set $\mathcal{N}$.*

(1) *An unsigned term t occurs in $n \in \mathcal{N}$ iff $t \sqsubset term(n)$;*
(2) *Suppose I is a set of unsigned terms. The node $n \in \mathcal{N}$ is an* entry point *for I iff $term(n) = +t$ for some $t \in I$, and whenever $n' \Rightarrow n$, $term(n') \notin I$;*
(3) *An unsigned term t* originates *on $n \in \mathcal{N}$ iff n is an entry point for the set $I = \{t' : t \sqsubset t'\}$;*
(4) *An unsigned term t is* uniquely originating *in a set of nodes $S \subset \mathcal{N}$ iff there is a unique $n \in S$ such that t originates on n. The term t is non-originating in $S \subset \mathcal{N}$ iff there is no $n \in S$ such that t originates on n.*

The atomic actions available to the penetrator (this term usually designates the intruder in the strand space theory) are encoded in a set of penetrator traces. They summarize his ability to discard messages, generate well-known messages, piece messages together, and apply cryptographic operations using keys that become available to him. A protocol attack typically requires hooking together several of these atomic actions.

The actions available to the penetrator are relative to the set of keys that the penetrator initially knows. We encode this in a parameter, the set of penetrator keys K_I.

Definition 3.3 *A penetrator trace relative to* K_I *is one of the following:*

- M_t*: Text message:* $\langle +t \rangle$ *where* $t \in \mathsf{T}$*;*
- K_K*: Key:* $\langle +K \rangle$ *where* $K \in \mathsf{K}_I$*;*
- $\mathsf{C}_{g,h}$*: Concatenation:* $\langle -g, -h, +g, h \rangle$*;*
- $\mathsf{Sep}_{g,h}$*: Separation:* $\langle -g, h, +g, +h \rangle$*;*
- $\mathsf{Sig}_{h,K}$*: Signature:* $\langle -K, -h, +\{\!|h|\!\}_K \rangle$*;*
- $\mathsf{Rec}_{h,K}$*: Recovery:* $\langle -K^{-1}, -\{\!|h|\!\}_K, +h \rangle$*;*

$\mathcal{P}_\Sigma$ is the set of all strands $s \in \Sigma$ such that $tr(s)$ is a penetrator trace.

A strand $s \in \Sigma$ is a *penetrator strand* if s belongs to $\mathcal{P}_\Sigma$, and a node is a *penetrator node* if the strand it lies on is a penetrator strand. Otherwise, we will call it a *non-penetrator* or *regular strand* or *node*.

3.2. Authentication Tests

Authentication tests provide a convenient method for establishing authentication properties of security protocols.

Suppose a principal in a cryptographic protocol creates and transmits a message containing a new value v, later receiving v back in a different cryptographic context. He can conclude that some principal possessing the relevant key K has received and transformed the message in which v was emitted. If the penetrator does not know K, he cannot have achieved this transformation, and it must therefore be a regular strand. A transforming edge is the action of changing the cryptographic form in which such a value v occurs. The authentication tests give sufficient conditions for transforming edges being the work of regular principals.

In order to write this more precisely, we introduce some new definitions:

Definition 3.4

(1) *If* $\mathcal{K} \subset \mathsf{K}$*, then* $t_0 \sqsubset_{\mathcal{K}} t$ *if* t *is an element of the smallest set containing* t_0 *and closed under encryption with* $K \in \mathcal{K}$ *and concatenation with arbitrary terms* t_1*;*

(2) *A term* t *is* simple *if it is not of the form* g, h*;*

(3) *A term* t_0 *is a* component *of* t*, written* $\boxed{t_0} \sqsubset t$ *if* t_0 *is simple and* $t_0 \sqsubset_\emptyset t$*;*

(4) *A term* t_0 *is a* new *component of a node* n *if* $\boxed{t_0} \sqsubset term(n)$*, and whenever* $m \Rightarrow^+ n$ *it is not the case that* $\boxed{t_0} \sqsubset term(m)$*;*

(5) *An edge* $n_1 \Rightarrow^+ n_2$ *is a* transforming edge *for* $a \in \mathsf{A}$ *if* n_1 *is negative and* n_2 *positive,* $a \sqsubset term(n_1)$*, and there is a new component* t_2 *of* n_2 *such that* $a \sqsubset t_2$*.*

We can now define a first authentication test:

Definition 3.5 *We say that* $n_0 \Rightarrow^+ n_1$ *is an* incoming test edge *for* a *in* $t_1 = \{\!|h|\!\}_K$ *if:*

- a *originates uniquely on* n_0*, and* $t_1 = \{\!|h|\!\}_K \not\sqsubset term(n_0)$*;*
- $a \sqsubset t_1$*, and* $t_1 \sqsubset term(n_1)$*;*
- $K \notin \mathsf{K}_I$*.*

The presence of an incoming test edge gives us the following guarantee:

Authentication Test 1 Let $\mathcal{C}$ be a bundle with $n_1 \in \mathcal{C}$, and let $n_0 \Rightarrow^+ n_1$ be an incoming test edge for a in $t = \{\!|h|\!\}_K$. Then there exist regular nodes $m_0, m_1 \in \mathcal{C}$ such that t is a component of m_1 and $m_0 \Rightarrow^+ m_1$ is a transforming edge for a. Furthermore, $n_0 \prec m_0 \prec m_1 \prec n_1$.

We will exploit another authentication test:

Definition 3.6 *A negative node* n *is an* unsolicited test *for* a *if:*

- $a = \boxed{\{\!|h|\!\}_K}$ *is received on* n*;*
- $K \notin \mathsf{K}_I$*.*

Unsolicited tests provide us the following guarantee:

Authentication Test 2 Let $\mathcal{C}$ be a bundle with $n \in \mathcal{C}$, and let n be an unsolicited test for $a = \{\!|h|\!\}_K$. Then there exists a positive regular node $m \in \mathcal{C}$ such that a is a component of m. Furthermore, $m \prec n$.

These two tests will be intensively exploited in order to prove the authentication properties of our protocol.

3.3. Recency

Our first security goal is an authentication property, and we just presented a way to verify this kind of goals. Our second security goal, the resistance to known session-secret attacks, is guaranteed if we are able to prove that group keys are built from recent contributions only, which are therefore independent of any old, maybe compromised session-secret.

Regular strands provide a way to measure recency. Indeed, implementers may always ensure that a local protocol run will timeout long before cryptanalysis could have succeeded. Thus, a principal engaged in

a regular strand knows that an event is recent if it happened after an earlier event on the same strand.

Definition 3.7 *A node n is* recent *for a regular node m_1 in a bundle $\mathcal{C}$ if there is a regular node $m_0 \in \mathcal{C}$ such that $m_0 \Rightarrow^+ m_1$ and $m_0 \preceq_{\mathcal{C}} n \prec_{\mathcal{C}} m_1$.*

This definition shows us that incoming tests entail recency: if $n_0 \Rightarrow^+ n_1$ is an incoming test edge, and if $m_0 \Rightarrow^+ m_1$ is the corresponding transforming edge, then m_0 and m_1 are recent for n_1.

In some cases, we need a more inclusive notion of recency:

Definition 3.8 *A node n is 1-recent for m_1 if n is recent for m_1 as in Definition 3.7. A node n is $i+1$-recent for m_1 if there exists a node m_0 such that n is i-recent for m_0 and m_0 is recent for m_1.*

If n is i-recent for m, then there are i strands, each overlapping a portion of the preceding one. From beginning to end, at most i times the timeout for a single regular strand can have elapsed.

Equipped with the authentication test machinery, we may now turn to the design of our protocol.

4. Construction of a New Protocol

4.1. Introduction

The starting point of our protocol is the GDH.2 protocol [73]. This protocol allows a group of principals to generate a key whose confidentiality is guaranteed in the presence of a passive attacker, i.e. an attacker who is only able to eavesdrop messages.

Following the design process suggested in [33], we now introduce authentication tests in the GDH.2 protocol in order to insure the security goals defined in Section 2.

4.2. Modelling of the GDH.2 Protocol

At first, we consider a group of 3 members only; the transposition to a group of n members will be provided further, as well as a proof of the correctness of the resulting protocol.

The protocol we have to design perfectly fits the classical strand spaces formalism, except that we have to deal with elements of a cyclic group $\mathcal{G}$. However, for our purpose, we do not need to take into account the particular properties of exponentiation: we would just like to authenticate terms and learn from which elements they are built.

A simple way to take these properties into account is as follows. We define two public values: a symmetric key $1 \in \mathsf{K}$ and a key $K_\alpha \in \mathsf{K}$

with no known inverse. We can then model α as a simplified writing for $\{\!|1|\!\}_{K_\alpha}$, and the exponentiation of a term h with a random value $r \in \mathsf{K}$ as a notation for $\{\!|h, r|\!\}_{K_\alpha}$. So, for instance, $\alpha^{r_1 r_2}$ will be considered as a concise way to write $\{\!|\{\!|\{\!|1|\!\}_{K_\alpha}, r_1|\!\}_{K_\alpha}, r_2|\!\}_{K_\alpha}$. This way of modelling the elements of $\mathcal{G}$ presents the advantage that it makes it impossible to learn r_1 or r_2 from $\alpha^{r_1 r_2}$ and keeps the intuition behind the subterm relation: with our notations, $r_1 \sqsubset \alpha^{r_1 r_2}$. However, when adopting this way of modelling exponentiation, $\alpha^{r_1 r_2} \neq \alpha^{r_2 r_1}$, what will impose us a particular treatment for the secrecy property.

A strand space representing a session of the GDH.2 protocol with three participants is represented in Fig. 3.1.

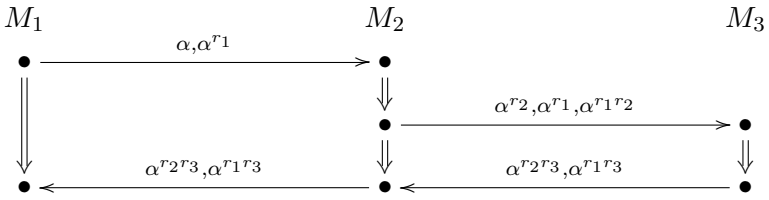


FIGURE 3.1. A run of the GDH.2 protocol with 3 participants.

When a participant receives a sequence of elements of $\mathcal{G}$, he typically cannot inspect it in order to see how it is built: when receiving α^{r_1}, M_2 cannot extract r_1 from this value. So, we will use a variable of form $g_i[j]$ to denote a sequence g_i of j elements of $\mathcal{G}$ and define $(g_i)_j$ as the j-th element of g_i. The roles in the protocol described above can then be defined through three parametric strands:

$$\begin{array}{rcl} SM_1[r_1, g_3[2]] & = & \langle +\alpha, \alpha^{r_1} \quad - g_3[2] \rangle \\ SM_2[g_1[2], r_2, g_3[2]] & = & \langle -g_1[2] \quad + (g_1)_1^{r_2}, (g_1)_2, (g_1)_2^{r_2} \quad - g_3[2] \rangle \\ SM_3[g_2[3], r_3] & = & \langle -g_2[3] \quad + (g_2)_1^{r_3} (g_2)_2^{r_3} \rangle \end{array}$$

where we assume that the values r_i are uniquely originating.

As we said above, this protocol is not intended to provide any authentication guarantee: a group member cannot have any confidence in the fact that the key he is computing at the end of the protocol execution is only known by the other expected group members since no identity related information is transmitted.

4.3. Authentication Services

We now try to transform the A-GDH.2 group key agreement protocol into an AGKAP. In order to be able to use the authentication tests defined above, we need to use some keys that are kept out of reach of the intruder. Since the messages exchanged during a GDH.2 protocol

session may be public, we choose to use signature schemes. We will assume that $S_i \notin \mathsf{K}_{\mathcal{P}}$ denotes the signing key of M_i, while S_i^{-1} is public.

Signing each flow provides unsolicited tests, which can be used to guarantee the regular origination of the random values exploited to build the exchanged messages. A session of the corresponding protocol is represented in Fig. 3.2.

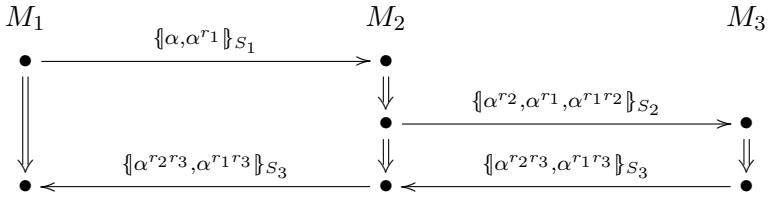


FIGURE 3.2. GDH.2 protocol run with all messages signed.

If we look at the strand executed by M_3, we observe that the message received on its first node is an unsolicited test for $\{\!|\alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1r_2}|\!\}_{S_2}$. This signed component must have been generated by M_2 since he is the only user who knows S_2. Furthermore, the length of this message insures that this component has been generated on a strand corresponding to the role of the second member of a group.

However, this message contains no information about the user who plays the role of the first group member. This information can be transmitted by adding the identifier M_1 into the message M_2 sends to M_3, which becomes $\{\!|M_1, \alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1r_2}|\!\}_{S_2}$. A similar consideration about the authentication of the final broadcast leads us to add the identifiers of the first two group members into this message, which becomes $\{\!|M_1, M_2, \alpha^{r_2r_3}, \alpha^{r_1r_3}|\!\}_{S_3}$. A typical run of the resulting protocol is represented in Fig. 3.3.

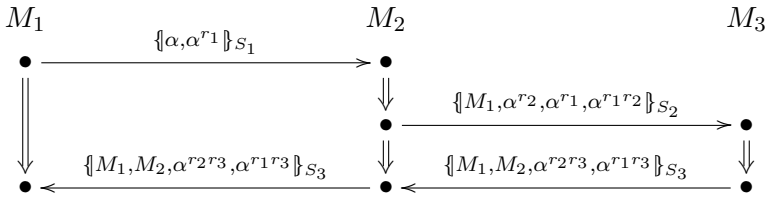


FIGURE 3.3. GDH.2 protocol run with signatures and identifiers.

Consider an execution of this protocol. When receiving the final broadcast, M_1 selects the value $\alpha^{r_2r_3}$ to compute his view of the group key. From the key used to build this broadcast, M_1 knows that $\alpha^{r_2r_3}$

has been computed by M_3. From the identifiers M_1 and M_2, he knows that the term from which M_3 computed $\alpha^{r_2 r_3}$ was received from M_2, who computed it from a value received from M_1.

So, at this point, each user can deduce who contributed to generate the messages he receives. This allows users to know that, when they compute a group key, it has only been contributed to by well defined users. This is stronger than what was required by the implicit key authentication property. It will however be useful to verify that this group key is never disclosed.

4.4. Secrecy

We have to verify that our protocol never discloses the group key, which, in our model, will be computed as $\alpha^{r_2 r_3 r_1}$, $\alpha^{r_1 r_3 r_2}$ and $\alpha^{r_1 r_2 r_3}$ by M_1, M_2 and M_3 respectively.

So, we have to prove that the term $\alpha^{r_a r_b r_c}$ where $\{r_a, r_b, r_c\}$ is a permutation of $\{r_1, r_2, r_3\}$ can never be exchanged on a readable form.

We provide the main ideas behind this verification in the case of the key computed by M_3. Let $\mathcal{C}$ be a bundle containing the strand executed by M_3 in Fig. 3.3 (we refer to this strand as s_3) and suppose a node $n \in \mathcal{C}$ to be a $\prec$-minimal node such that $\alpha^{r_a r_b r_c} \sqsubset term(n)$.

By inspection, this node cannot be a penetrator node: the only minimal penetrator node on which $\alpha^{r_a r_b r_c}$ could occur is the third of a $\mathsf{Sig}_{\{\alpha^{r_a r_b}, r_c\}, K_\alpha}$ strand, but r_c is uniquely originating on a regular strand and is only communicated encrypted through a key with no known inverse, and it can therefore not be disclosed.

So, n must be a regular node. By inspection, and from the minimality assumption of n, the occurrence of $\alpha^{r_a r_b r_c}$ in $term(n)$ must be the result of the exponentiation of a previously received value with r_c. But the unique origination of r_c and our authentication properties guarantee us that:

- the strand on which r_1 originates only emits α and α^{r_1};
- the strand on which r_2 originates only emits α^{r_2}, α^{r_1} and $\alpha^{r_1 r_2}$;
- the strand on which r_3 originates only emits $\alpha^{r_2 r_3}$ and $\alpha^{r_1 r_3}$.

So, none of the three conceivable values of r_c can be the right one. The node n defined above does therefore not exist and $\alpha^{r_a r_b r_c}$ is kept secret.

A similar reasoning can be carried out for the other two group members and the implicit key authentication is then guaranteed. However, our protocol does not provide any information about the freshness of the elements that are used to build the group key. Such information would nevertheless be useful if we want our protocol to be resistant to known session-secret attacks.

4.5. Recency

We will now adapt our protocol in order to be able to prove that, when a user computes a group key at the end of a session of our protocol, all contributions to this key uniquely originate on recent nodes.

If we look at the two authentication tests above, we can observe that the incoming test is the only one which can be used to obtain such guarantees: the unsolicited test does not provide any lower $\prec$-bound on regular nodes.

As in the previous section, we first look at the way an incoming test edge can be obtained for M_3. In order to contain such an edge, SM_3 must contain two nodes n, n' such that n is a positive node and $n \Rightarrow^+ n'$, which is not the case up to now. We will therefore transform our protocol by adding a new node at the beginning of the SM_1 and SM_3 strands, which will allow the transmission of a nonce N uniquely originating on strands of type SM_3. This nonce will be forwarded back to s_3 through the up-flows of the protocol as represented in Fig. 3.4.

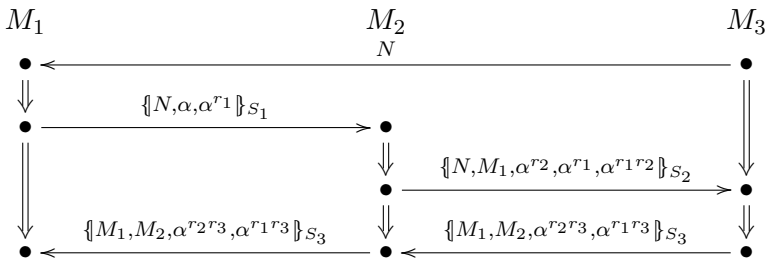


FIGURE 3.4. GDH.2 protocol run with signatures, identifiers and a nonce.

When receiving $\{\!|N, M_1, \alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1r_2}|\!\}_{S_2}$, M_3 knows that this component has been generated by M_2 after the sending of N since it contains this random value. Furthermore, M_2 must have built this message from a message sent by M_1, this message also containing the nonce N. So, M_3 knows that r_1 and r_2 are fresh values (and not replay of old ones) since both are uniquely originating after N was emitted.

We can now turn to the way recency can be entailed for M_1 and M_2. The corresponding two roles already contain an incoming test edge: M_1, for instance, sends a term containing r_1 to M_2 and receives this term back in a different cryptographic context in the final broadcast. However, M_1 is not able to check whether $r_1 \sqsubset \{\!|M_1, M_2, \alpha^{r_2r_3}, \alpha^{r_1r_3}|\!\}_{S_3}$: it is encrypted through K_α together with r_3. A solution to this problem is to include a new element in the broadcast, element that will be used by M_1 to verify whether this broadcast has been generated after the

emission of r_1. Including the group key M_1 and M_2 will compute in a hidden form is a convenient way to achieve this. This form could be a hash of $\alpha^{r_1 r_2 r_3}$, denoted $\mathcal{H}(\alpha^{r_1 r_2 r_3})$: if the hash of a function of r_1 is present in a message, then r_1 has been used to build this hash. $\mathcal{H}(g)$ can be seen as $\{\!|g|\!\}_{K_{\mathcal{H}}}$ where $K_{\mathcal{H}}$ is a public key with no known inverse. If we add this hash into the last message of the protocol, a typical run becomes as represented in Fig. 3.5.

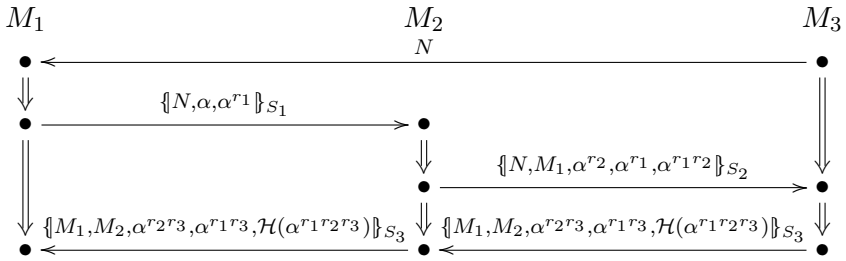


FIGURE 3.5. AT-GDH protocol run with 3 participants

At the end of a session of this protocol, M_1 and M_2 can verify that the broadcast they receive has been sent recently since it must have been constituted by M_3 after the sending of r_1 and r_2. Furthermore, M_1 and M_2 know that M_3 generated this broadcast from values he knows to be more recent than the emission of N. So, M_1 and M_2 know that the key they are computing at the end of a protocol session is made up of contributions generated at most two time-out periods before the reception of the broadcast.

5. Correctness of the AT-GDH Protocol

5.1. The AT-GDH Protocol

In the previous section, we used authentication tests to design a new AGKAP with three participants. We now consider the n-party case and prove its correctness.

Our three-party protocol can be easily extended to the general n-party case. The extension we suggest is as follows:

- As previously, the last group member, M_n, sends a nonce N to M_1;
- M_1 sends the same message as in our three-party protocol;
- Each user M_i $(1 < i \leq n)$ receives a message signed by M_{i-1} containing:
  - the nonce N;

  - the concatenation of the identifiers of the $i-2$ first group members;
  - the elements of $\mathcal{G}$ corresponding to those that M_i should receive during the up-flow of an execution of the GDH.2 protocol;
- After having received this message, each user M_i $(1 < i < n)$ sends to M_{i+1} a message generated accordingly;
- M_n receives the message M_{n-1} sent him, checks the signature, the nonce N and the $n-2$ identifiers, computes the group key $\alpha^{r_1 \cdots r_n}$ and generates the message he will broadcast to the $n-1$ first group members. This last signed message will contain:
  - the concatenation of the identifiers of the $n-1$ first group members;
  - the elements of $\mathcal{G}$ corresponding to those that M_n should send in the broadcast of an execution of the GDH.2 protocol;
  - the hash of the group key;
- Finally, the $n-1$ first group members check the signature, the $n-1$ identifiers, computes the group key $\alpha^{r_1 \cdots r_n}$, and check that the hash of this key was present in the message.

If we adopt the notation $\mathsf{M}_{<i}$ to denote the (ordered) set $\{M_1, \ldots, M_{i-1}\}$ and $(g)^r_{<i}$ to denote $(g)^r_1, \ldots, (g)^r_{i-1}$, the n parametric strands corresponding to the AT-GDH protocol with n parties are the following:

$$
\begin{aligned}
&SM_1[N, \mathsf{M}, r_1, g_n[n-1]] = \\
&\qquad \langle -N \quad + \{\!|N, \alpha, \alpha^{r_1}|\!\}_{S_1} \\
&\qquad\qquad\qquad\qquad -\{\!|\mathsf{M}_{<n}, g_n[n-1], \mathcal{H}((g_n)^{r_1}_1)|\!\}_{S_n} \rangle \\
&SM_i[N, \mathsf{M}, g_{i-1}[i], r_i, g_n[n-1]] = \\
&\qquad \langle -\{\!|N, g_{i-1}[i]|\!\}_{S_{i-1}} \quad + \{\!|N, \mathsf{M}_{<i}, \quad (g_{i-1})^{r_i}_{<i}, (g_{i-1})_i, (g_{i-1})^{r_i}_i|\!\}_{S_i} \\
&\qquad\qquad\qquad\qquad -\{\!|\mathsf{M}_{<n}, g_n[n-1], \mathcal{H}((g_n)^{r_i}_i)|\!\}_{S_n} \rangle \\
&SM_n[N, \mathsf{M}, g_{n-1}[n], r_n] = \\
&\qquad \langle +N \quad - \{\!|N, \mathsf{M}_{<n-1}, g_{n-1}[n]|\!\}_{S_{n-1}} \\
&\qquad\qquad\qquad\qquad +\{\!|\mathsf{M}_{<n}, (g_{n-1})^{r_n}_{<n}, \mathcal{H}((g_{n-1})^{r_n}_n)|\!\}_{S_n} \rangle
\end{aligned}
$$

It may be verified that, when $n = 3$, these strands correspond to those represented in Fig. 3.5.

We now verify the security properties defined in Section 2 for this protocol. In the rest of this section, we assume that:

- $\mathcal{C}$ is a AT-GDH bundle;
- M contains n group members, where $n \geq 2$;
- $\mathsf{K}_{\mathcal{P}} = \mathsf{K} \backslash (\{S_1, \ldots S_n\} \cup K_\alpha^{-1} \cup K_{\mathcal{H}}^{-1})$;

- r_i $(1 \leq i \leq n)$ uniquely originates on a SM_i strand.

Furthermore, when considering parametric strands, we write $*$ in particular argument positions to indicate a union. For instance,

$$SM_1[N, \mathsf{M}, *, g_n[n-1]] = \bigcup_{r_1} SM_1[N, \mathsf{M}, r_1, g_n[n-1]]$$

is the set of all SM_1 strands involving a nonce N, the group of users M, a given sequence of $n-1$ elements of G, with any random value r_1. We also use $**$ to indicate that multiple adjacent arguments have been projected, writing e.g. $SM_1[N, M_1, M_2, M_3, **]$ for $SM_1[N, M_1, M_2, M_3, *, *, *]$.

5.2. Achieving Implicit Key Authentication

As we described above, the implicit key authentication goal is achieved in two steps: we first prove authentication properties about the keying material group members are using, then we show that the key they are computing is kept secret.

We first prove authentication properties for the last group member.

Proposition 3.1 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s_n \in SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$ of $\mathcal{C}$-length ≥ 2. Then $(g_{n-1})_n = \alpha^{r_1 \ldots r_{n-1}}$ where r_i is uniquely originating on a SM_i-strand executed by M_i.*

Proof. Suppose $s_n \in SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$. $\langle s_n, 2\rangle$ is an unsolicited test for $t = \{\!|N, \mathsf{M}_{<n-1}, g_{n-1}[n]|\!\}_{S_{n-1}}$. Therefore, t is a component of a positive regular node. The only positive regular node which may contain t is a node $\langle s_{n-1}, 2\rangle$ where $s_{n-1} \in SM_{n-1}[\mathsf{M}_{<n}, *, g_{n-2}[n-1], r_{n-1}, **]$. So, $\langle s_{n-1}, 1\rangle$ exists, and this node is an unsolicited test for $t' = \{\!|N, \mathsf{M}_{<n-2}, g_{n-2}[n-1]|\!\}_{S_{n-2}}$. Furthermore, from the definition of the SM_{n-1}-strands, $(g_{n-1})_n = (g_{n-2})_{n-1}^{r_{n-1}}$. We may now proceed recursively on the unsolicited tests until we reach a strand s_1 of type $SM_1[N, M_1, **, r_1, **]$. By identifying the terms, we may then verify that $(g_{n-1})_n = \alpha^{r_1 \ldots r_{n-1}}$ where r_i is uniquely originating on a SM_i-strand executed by M_i. ∎

Similar results can be obtained for the other group members.

Proposition 3.2 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s_i \in SM_i[N, \mathsf{M}, g_{i-1}[i], r_i, g_n[n-1]]$ of $\mathcal{C}$-length $= 3$ (we assume $1 \leq i < n$ and $g_0 = \emptyset$). Then $(g_n)_i = \alpha^{r_1 \ldots r_{i-1} r_{i+1} \ldots r_n}$ where r_j is uniquely originating on a SM_j-strand executed by M_j.*

Proof. $\langle s_i, 3\rangle$ is an unsolicited test for $t = \{\!|\mathsf{M}_{<n}, g_n[n-1], \mathcal{H}((g_n)_i^{r_i})|\!\}_{S_n}$. Therefore, t is a component of a positive regular node. The only positive regular node that may contain t is a node $\langle s_n, 3\rangle$ where $s_n \in$

$SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$. So, $\langle s_3, 2\rangle$ exists, and the arguments of Proposition 3.1 can be re-used to prove the existence of nodes that ensure the correctness of this proposition. ∎

Authentication properties having been established, we now turn to secrecy properties. Given a session of the AT-GDH protocol executed by a group M of cardinality n such that M_i's contribution to the key is r_i, the intruder comes into possession of the group key if a term of the form $\alpha^{s_1 \cdots s_n}$ where $\{s_1, \ldots, s_n\}$ is a permutation of $\{r_1, \ldots, r_n\}$ is transmitted without being protected through a key the intruder ignores. We will prove that no node containing such a value can exist in $\mathcal{C}$. As for the authentication properties, we are giving a complete proof of the confidentiality of the key computed by M_n; the transposition to the other cases being straightforward.

Proposition 3.3 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s \in SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$ of $\mathcal{C}$-length $= 3$. Then, if $(g_{n-1})_n = \alpha^{r_1 \cdots r_{n-1}}$, there is no node $n \in \mathcal{C}$ such that $term(n) = \alpha^{s_1 \cdots s_n}$ where $\{s_1, \ldots, s_n\}$ is a permutation of $\{r_1, \ldots, r_n\}$.*

Proof. Let $\mathcal{N} = \{n \in \mathcal{C} : \alpha^{s_1 \cdots s_n} \sqsubset term(n)\}$ where $\{s_1, \ldots, s_n\}$ is a permutation of $\{r_1, \ldots, r_n\}$. $\mathcal{N}$ must contain $\prec$-minimal nodes. Let m be such a node.

At first, suppose m lies on a penetrator strand. We successively consider the different possible cases.

- m belongs to a M_t-strand: this is not possible since $\alpha^{s_1 \cdots s_n} \notin \mathsf{T}$;
- m belongs to a K_K-strand: this is not possible since $\alpha^{s_1 \cdots s_n} \notin \mathsf{K}$;
- m belongs to a $\mathsf{C}_{g,h}$-strand: this is not possible since m must be a positive node and if $\alpha^{s_1 \cdots s_n} \sqsubset g, h$ then $\alpha^{s_1 \cdots s_n} \sqsubset g$ or $\alpha^{s_1 \cdots s_n} \sqsubset h$;
- m belongs to a $\mathsf{Sep}_{g,h}$-strand: this is not possible since m must be a positive node and if $\alpha^{s_1 \cdots s_n} \sqsubset g$ or $\alpha^{s_1 \cdots s_n} \sqsubset h$ then $\alpha^{s_1 \cdots s_n} \sqsubset g, h$;
- m belongs to a $\mathsf{Rec}_{h,K}$-strand: this is not possible since m must be a positive node and if $\alpha^{s_1 \cdots s_n} \sqsubset h$ then $\alpha^{s_1 \cdots s_n} \sqsubset \{\!|h|\!\}_K$;

Finally, m must belong to a $\mathsf{Sig}_{h,K}$-strand and this strand must have the following form:

$$\langle -K_\alpha \quad - \alpha^{s_1 \cdots s_{n-1}}, s_n \quad + \alpha^{s_1 \cdots s_n} \rangle$$

This is however impossible because s_n uniquely originates on a regular strand and all occurrences of s_n are encrypted through the key K_α which has no known inverse. So, s_n cannot be exposed in a readable form (or, in the words of [32] page 245, s_n is a safe key since $s_n \in \mathsf{S}_1$).

So, m cannot lie on a penetrator strand. Suppose now that m lies on a regular strand.

By inspection, if $\alpha^{s_1 \cdots s_n} \sqsubset m$, $\alpha^{s_1 \cdots s_n}$ must be a subterm of an emitted element of $\mathcal{G}$. Assume $\alpha^{s_1 \cdots s_n} \sqsubset \alpha^x \sqsubset term(m)$ where α^x is not contained in an encryption through K_α. Inspecting the regular strands shows that α^x can be emitted in three ways. First, it can be a copy of a previously received element of $\mathcal{G}$. But this is not possible given that m must be $\prec$-minimal. Second, it can be emitted in the broadcast, encrypted through the $K_\mathcal{H}$ key. But this cannot lead to the compromise of $\alpha^{s_1 \cdots s_n}$ since $K_\mathcal{H}$ has no known inverse. Finally, α^x can be the result of the exponentiation of a received value with a random contribution r_i. The $\prec$-minimality of m and the observation that each regular strand performs exponentiation with one only random value involve that $\alpha^x = \alpha^{s_1 \cdots s_n}$ and that $r_i = s_n$. But the unique origination of r_i on a $SM_i[N, \mathsf{M}_{\leq i}, **, r_i, **]$-strand established in Propositions 3.1 and 3.2 also involves that α^x cannot contain n nested exponentiations (or encryptions with K_α).

So, the node m can only be the third of the s strand, and $\alpha^{s_1 \cdots s_n}$ can only appear encrypted through the key $K_\mathcal{H}$. But, since $K_\mathcal{H}$ has no known inverse, $\alpha^{s_1 \cdots s_n}$ is kept out of reach of the intruder and the node n described in the wording of this proposition cannot exist. ∎

A similar proof can be carried out for the $n-1$ other group members.

Proposition 3.4 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s \in SM_i[N, \mathsf{M}, g_{i-1}[i], r_i, g_n[n-1]]$ of $\mathcal{C}$-length $= 3$ where $i < n$ and g_0 is empty. Then, if $(g_n)_i = \alpha^{r_1 \cdots r_{i-1} r_{i+1} \cdots r_n}$, there is no node $n \in \mathcal{C}$ such that $term(n) = \alpha^{s_1 \cdots s_n}$ where $\{s_1, \ldots, s_n\}$ is a permutation of $\{r_1, \ldots, r_n\}$.*

This achieves our verification of the implicit key authentication property: we first authenticated the group key computed by each group member, then we proved that these keys cannot be compromised.

We in fact proved something stronger: all group members M_i ($1 \leq i < n$) have the guarantee that M_n knows the key they computed. This corresponds to explicit key authentication, but only for one user, and in one direction: the group member M_i ($1 \leq j \leq n$) does not know whether M_j ($1 \leq j < n$) shares a key with him.

We now turn to our second security property: the resistance to known session-secret attacks.

5.3. Achieving Resistance to Known Session-Secret Attacks

In the previous section, we proved that the key computed by a user executing a session of the AT-GDH protocol for a group of n users M is

a function of contributions $r_1, \ldots, r_n$ where r_i is uniquely originating on a SM_i-strand executed by M_i.

The resistance to known session-secret attacks can be guaranteed if we are able to prove that these contributions are recent: the compromise of old session-secrets would then not influence new sessions. This can be proved by using authentication tests.

As above, our first proposition concerns SM_n strands.

Proposition 3.5 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s_n \in SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$ of $\mathcal{C}$-length ≥ 2. Then, if $(g_{n-1})_n = \alpha^{r_1 \ldots r_{n-1}}$, then $r_1, \ldots, r_{n-1}$ originate on nodes recent for $\langle s_n, 2\rangle$.*

Proof. Consider the AT-GDH-bundle defined above. $\langle s_n, 2\rangle$ is an unsolicited test for $t = \{\!|N, \mathsf{M}_{<n-1}, g_{n-1}[n]|\!\}_{S_{n-1}}$. Then, there exist a positive regular node n such that t is a component of n. The only regular node corresponding to this description is the second node of a strand $s_{n-1} \in SM_{n-1}[N, \mathsf{M}_{<n}, *, g_{n-2}[n-1], r_{n-1}, *]$. So, from the properties of the unsolicited tests, $\langle s_{n-1}, 2\rangle \prec \langle s_n, 2\rangle$.

Since $\langle s_{n-1}, 2\rangle \in \mathcal{C}$, $\langle s_{n-1}, 1\rangle \in \mathcal{C}$ too, and this node constitutes an unsolicited test. By repeating the process we adopted for $\langle s_n, 2\rangle$, we can deduce the existence of $n-2$ other strands $s_i \in SM_i[N, \mathsf{M}_{\leq i}, g_{i-1}[i], r_i, g_n[n-1]]$ of $\mathcal{C}$-length ≥ 2 where $1 \leq i \leq n-2$ and g_0 is empty. These tests also guarantee that $\langle s_i, 2\rangle \prec \langle s_{i+1}, 2\rangle$ $(1 \leq i < n)$.

We would now like to prove that $\langle s_n, 1\rangle \prec \langle s_1, 2\rangle$. This can be deduced from the unique origination of the nonce N. Let $\mathcal{N}$ be the set of nodes in $\mathcal{C}$ such that for any $n \in \mathcal{N}$, $N \sqsubset term(n)$. This set is clearly not empty; so, it has $\prec_{\mathcal{C}}$-minimal members. The sign of $\langle s_1, 1\rangle \in \mathcal{N}$ is negative, so there exists a node $m \in \mathcal{N} : n \prec \langle s_1, 1\rangle$. If we assume m to be minimal, its sign is positive and $\forall m' : m' \Rightarrow^+ m$, $N \not\sqsubset term(m')$. Then N is originating on m. But since N is uniquely originating on $\langle s_n, 1\rangle$, $m = \langle s_n, 1\rangle$. So, we proved that $\langle s_n, 1\rangle \prec \langle s_1, 2\rangle \prec \cdots \prec \langle s_n, 2\rangle$, which implies that $r_1, \ldots, r_{n-1}$ are originating on nodes that are recent for $\langle s_n, 2\rangle$. ∎

This result can be used to establish recency properties for the other types of strands.

Proposition 3.6 *Suppose a AT-GDH-bundle $\mathcal{C}$ containing a strand $s_i \in SM_i[N, \mathsf{M}, g_{i-1}[i], r_i, g_n[n-1]]$ of $\mathcal{C}$-length $= 3$ where $i < n$ and g_0 is empty. Then, if $(g_n)_i = \alpha^{r_1 \ldots r_{i-1} r_{i+1} \ldots r_n}$, then $r_1, \ldots, r_n$ originate on nodes recent for $\langle s_i, 3\rangle$.*

Proof. Consider the AT-GDH-bundle defined above. $\langle s_i, 2\rangle \Rightarrow \langle s_i, 3\rangle$ constitutes an incoming test for r_i in $t = \{\!|\mathsf{M}_{<n}, g_n[n-1], \mathcal{H}((g_n)_i^{r_i})|\!\}_{S_n}$.

So there exist regular nodes n, n' such that t is a component of n' and $n \Rightarrow^+ n'$ is a transforming edge for r_i. By inspection, n and n' can only be the second and third nodes of a strand $s_n \in SM_n[N, \mathsf{M}, g_{n-1}[n], r_n]$. Furthermore, since N is uniquely originating on SM_n strands, there is only one strand which can match these parameters. This implies that $\langle s_i, 2\rangle \prec \langle s_n, 2\rangle \prec \langle s_n, 3\rangle \prec \langle s_i, 3\rangle$, and the node $\langle s_n, 3\rangle$ on which r_n originates is therefore recent for $\langle s_i, 3\rangle$. Furthermore, $\langle s_n, 2\rangle$ is also recent for $\langle s_i, 3\rangle$, and we proved in Proposition 3.5 that $r_1, \ldots r_{n-1}$ are originating on nodes recent with respect to $\langle s_n, 2\rangle$. So, all these random contributions are originating on nodes at most 2-recent for $\langle s_i, 3\rangle$. ∎

In fact, a tighter analysis would have shown that the random contributions $r_{i+1}, \ldots, r_n$ were originating on nodes 1-recent for $\langle s_i, 3\rangle$, but this is not necessary for our purpose.

Finally, we have shown that, when a user computes a key in a group of honest participants, he can be sure that all contributions to this key were generated at most two timeout periods before this key computation, which keeps this key protected against known session-secret attacks. We can now turn to our last security property.

5.4. Achieving Individual Forward Secrecy

This property is trivially achieved for our protocol: the only long-term keys we are using are signing keys, so the compromise of these keys does not provide the penetrator any new information about values exchanged during past sessions of our protocol.

This concludes our establishment of the security goals of the AT-GDH protocol.

6. Comparison with the AKE1 Protocol

6.1. The AKE1 Protocol

The AT-GDH protocol has many common points with another AGKAP recently proposed by Bresson, Chevassut and Pointcheval: the AKE1 protocol [13].

As the AT-GDH protocol, the AKE1 protocol finds its roots in the GDH.2 protocol [73]. Authentication services are added on this basic structure in order to make the protocol exploitable in the presence of an active attacker. Keeping our previous notations, a typical run of the AKE1 protocol executed by three members is represented if Fig. 3.6.

Even though this protocol is very similar to the AT-GDH one, several important differences can be observed:

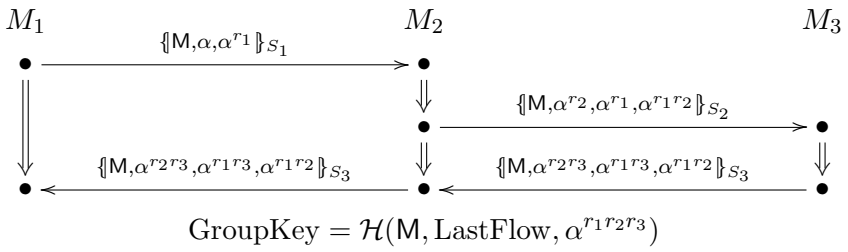


FIGURE 3.6. A run of the key-setup phase of the AKE1 protocol.

- The full group constitution M is included in each flow while our protocol only includes selected group members;
- There is no nonce included in the up-flow, nor hash in the broadcast;
- The group key is defined as the hash of the group constitution concatenated to the last message and $\alpha^{r_1 \cdots r_n}$.

Security proofs for this protocol have been established in the random oracle model, and a proof in the standard model of the security of a slightly different protocol has been established [14] (in this last protocol, the signatures are replaced by MAC's).

We now examine the reasons of these differences.

6.2. Inclusion of the Full Group Constitution

The AKE1 protocol includes the full group constitution M in each exchanged message, which seems to be redundant given our previous analysis, at least in the context of the security properties we considered.

In fact, in the models defined by Bresson & al. in [13, 14], the presence of identifiers in the signed messages is not required from a security point of view: it is assumed that only honest users can take part to sessions of the protocol, which implies that the penetrator cannot be a legitimate member of any group.

This assumption however keeps some plausible scenarios out of scope of their models. Imagine for instance a protocol AKE1′ defined as the AKE1 protocol except that the group constitution M is kept out of the signatures (i.e., the message $\{\!|\mathsf{M}, \alpha^{x_1}, \ldots \alpha^{x_i}|\!\}_{S_j}$ in the AKE1 protocol is transformed into $\mathsf{M}, \{\!|\alpha^{x_1}, \ldots \alpha^{x_i}|\!\}_{S_j}$). The security proof given in [13] remains valid for this protocol.

If we define $\mathsf{M_a}$ as $\{M_I, M_2, M_3\}$ where M_I is the intruder and $\mathsf{M_b}$ as $\{M_1, M_2, M_3\}$, the scenario represented in Fig. 3.7 appears to be problematic.

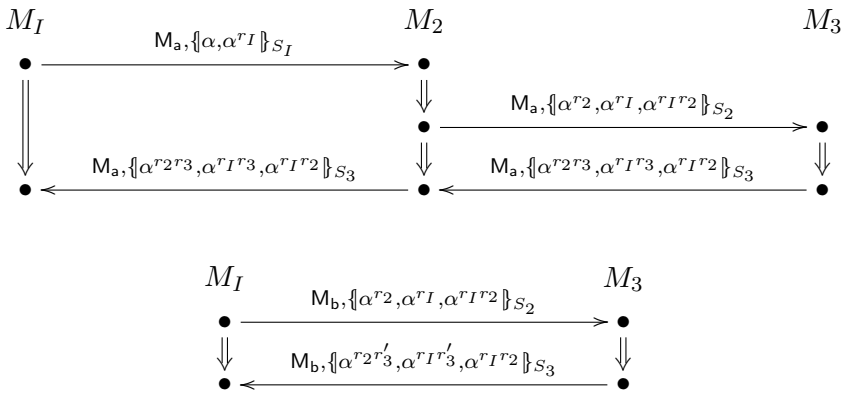


FIGURE 3.7. An attack against the AKE1′ protocol.

In this scenario, we consider two sessions of the AKE1′ protocol. In the first one, the intruder M_I is a legitimate group member whose contribution to the group key is r_I and whose long-term signing key is S_I. This session is executed as expected in the protocol definition. We then consider a second session of the protocol at the end of which M_3 is expecting to share a key with M_1 and M_2. However, the message M_3 receives is formed by the constitution of the second group $\mathsf{M_b}$ concatenated to the signed part of the message M_2 sent to M_3 during the first protocol session. This message has the structure M_3 expects, so he will respond with the message $\mathsf{M_b}, \{\!|\alpha^{r_2 r'_3}, \alpha^{r_I r'_3}, \alpha^{r_I r_2}|\!\}_{S_3}$ and compute $\alpha^{r_I r_2 r'_3}$ as group key, value that the intruder can compute from $\alpha^{r_2 r'_3}$.

We think this scenario shows the limits of the assumption stated in [13, 14].

6.3. Recency Properties

Contrary to the AT-GDH protocol, the AKE1 protocol does not include any way to measure the recency of the achieved messages. A consequence of this is the sensibility of this protocol to known session-secret attacks, as shown in Fig. 3.8.

In this scenario, the group $\mathsf{M} = \{M_1, M_2, M_3\}$ executes a first session of the protocol, session during which the contributions r_1, r_2 and r_3 are emitted. We assume the intruder eavesdrops the message $\{\!|\alpha, \alpha^{r_1}|\!\}_{S_1}$ emitted by M_1 and, and succeeds in obtaining r_1 (after several months of cryptanalytic work for instance).

Now, we assume the same group M executes the AKE1 protocol anew (maybe because the first group key they generated was old enough for cryptanalysis succeeding with non negligible probability), and the intruder replaces the message M_1 sends in this new session with the

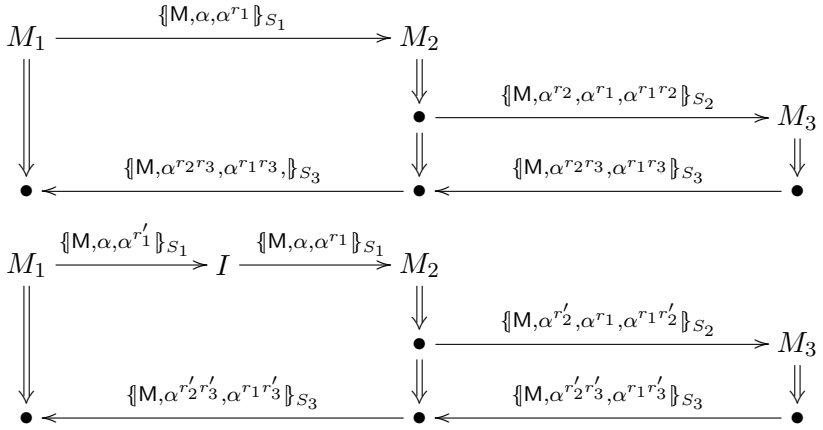


FIGURE 3.8. Sensibility of the AKE1 protocol to known session-secret attack

old one. M_2 and M_3 are generating fresh contributions r'_2 and r'_3 and the rest of the protocol is executed as expected. However, it may be observed that the group key M_2 and M_3 are computing at the end of this session is $\alpha^{r_1 r'_2 r'_3}$, value that the intruder can easily compute from the value $\alpha^{r'_2 r'_3}$ that M_3 broadcasted.

This kind of scenario is however not considered in [13, 14], so we cannot consider it an attack against this protocol. In [14], the authors however consider two kinds of corruptions: weak corruption in which some long-term signing keys can be compromised, and strong corruption in which signing keys as well as key contributions can be compromised. These corruption modes were also adopted in [69] for instance.

Our scenario in fact corresponds to a third type of corruption: the compromise of key contributions without compromising long-term signing keys. We think such a scenario is realistic in the practice: the security of the signing key and of the random generator are two independent things, as well as the forgery of a signature and the resolution of an instance of the discrete logarithm problem.

Furthermore, this scenario seems to have more problematic consequences than the compromise of a signing key. If M_1's signing key is compromised, the revocation of this key and the creation of a new one is sufficient to prevent problems in the future. Considering the compromise of r_1, we can imagine several solutions:

- imposing all group members to store the messages containing compromised values in order to prevent replays;
- imposing all group members (except the last one) to change their long-term signing key;

- impose M_1 to change his identifier.

Our first solution does not seem to be recommendable in the practice: managing such a library would be prohibitive. The second solution is very embarrassing: requiring M_2 to change his long-term signing key because M_1's contribution to a past session has been compromised does not seem to be admissible. Only changing M_1's signing key is however not sufficient: in the second session, the intruder would still be able to replace the message M_2 sends to M_3 with the message $\{\!|\mathsf{M}, \alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1 r_2}|\!\}_{S_2}$ sent during the first protocol session, which would result in M_3 computing the compromised value $\alpha^{r_1 r_2 r'_3}$ as group key. Finally, the solution of M_1 adopting a new identifier does not seem convenient.

We think that this scenario emphasizes the interest of including guarantees of the freshness of the key in AGKAP's.

6.4. Definition of the Group Key

If we look at the AKE1 protocol definition, we may observe that the group key is computed as $\mathcal{H}(\mathsf{M}, \text{LastFlow}, \alpha^{r_1 r_2 r_3})$, while our protocol simply uses $\alpha^{r_1 r_2 r_3}$.

A motivation for this choice is the requirement of key indistinguishability: the authors of the AKE1 protocol require that the intruder must be unable to distinguish a group key from a random value of the same length. Since an AKE1 protocol execution can be followed by an execution of a Mutual Authentication (MA) protocol [13] during which hashes of $\alpha^{r_1 r_2 r_3}$ and other public values are sent, $\alpha^{r_1 r_2 r_3}$ is distinguishable from a random sequence of bits. Adopting $\mathcal{H}(\mathsf{M}, \text{LastFlow}, \alpha^{r_1 r_2 r_3})$ as group key prevents this problem.

The AT-GDH protocol does not provide key indistinguishability: when asked whether a given value is the group key or not, the intruder can simply hash the key and compare it to the hash sent in the broadcast of the corresponding protocol session to be able to give a plausible answer.

The AT-GDH protocol can however be easily modified in order to meet this stronger notion of secrecy, by imposing to compute the group key as in the AKE1 protocol for instance. Another benefit of this way of computing the group key is to allow the confidentiality of the key to rely on the hardness of the Computational Diffie-Hellman problem (i.e. given α^x and α^y, compute α^{xy}) rather than on the hardness of the Decisional Diffie-Hellman problem.

6.5. Computational Considerations

The proofs of [13, 14] were established in a computational model: the authors of these papers are proving that an attacker who is able to break the security properties defined in their model can be exploited to solve at least one well-known computationally hard problem (e.g. to forge a signature or solve an instance of the DDH problem). Furthermore, since these proofs are providing exact relations between these problems (rather than asymptotical ones), information about the size of the security parameter to be adopted can be derived.

The model in which we proved the security of the AT-GDH protocol does not provide any information of this kind. It is however probable that arguments taken from [34] can be adapted to a slight modification of the AT-GDH protocol, what would allow us to obtain similar computational information.

At first, this adaptation would imply to transform our protocol in such a way that it would exploit Message Authentication Codes (MAC's) rather than a signature scheme: this would allow us to benefit from the properties of Carter-Wegman universal classes of hash functions [21] when considering computational aspects. A first information could then be obtained by measuring the probability for the adversary to succeed in forging a message for a given bundle. Adding this probability to the one of obtaining a group key by exploiting the Group Diffie-Hellman distribution would provide a result similar to the one obtained by Bresson & al. for the key authentication property [14].

Checking recency properties would require to take into account the probability of nonce clashes (which would allow the intruder to replay old messages), but also the probability of random contribution clashes which raises different problems since the adversary will not be able to check whether a clash occurred, except when these old contributions have been compromised or for selected group members (e.g. for the contributions r_1 and r_2 of M_1 and M_2 in the AT-GDH protocol since the values α^{r_1} and α^{r_2} are sent. On the other hand, r_3 and the other key contributions always occur exponentiated with other contributions, making it hard to check whether they have been already used in other sessions).

In a somewhat different direction, it would be also interesting to investigate how the work of J. Herzog on the modelling of the two-party Diffie-Hellman primitive in the strand space model [38] could be extended to protocols based on the group Diffie-Hellman primitive.

6.6. Efficiency Considerations

The AKE1, AT-GDH, and Cliques GDH protocols present similar efficiency characteristics in terms of required bandwidth, computational resources and number of rounds.

The bandwidth they require is quite modest: there are $n-1$ (or n for the AT-GDH protocol) messages exchanged during the upflow, and one broadcast (for a group of n members). By comparison, the AGKAP proposed by Katz and Yung in [40] (which is an extension of the Burmester-Desmedt protocol [19]) requires $3n$ broadcasts.

On the other side, GDH protocols require $\mathcal{O}(n)$ rounds while the Katz and Yung protocol requires only 3 rounds, and a GDH protocol requires $\mathcal{O}(n)$ modular exponentiations for each user, while the Katz and Yung protocol requires a constant number of modular exponentiations.

These costs imply that GDH protocols are not practical for groups larger than a hundred members [14]. A more detailed comparison of the efficiency of several group key agreement protocols can be found in [5].

7. Concluding Remarks

The conclusions of the first two chapters of this thesis were not really encouraging from a practical point of view: we pointed out attacks against well known AGKAP's and proved they could not be fixed without adopting new design rules. In this chapter, we decided to adopt different cryptographic primitives (namely a signature scheme and a hash function) and designed a protocol we called the AT-GDH protocol. Our design was motivated by security requirements rather than by routing efficiency, resistance to network failures [41, 42], or other such properties.

The AT-GDH protocol was designed in parallel with the AKE1 protocol proposed in [13]. The security of the AKE1 protocol has been established in a computational model, and its comparison with the AT-GDH protocol emphasizes the benefits and drawbacks of these two models families: the logical models allow us to reason finely about the content of the messages, and give justification of all message parts. Furthermore, they appear to be more suitable for reasoning about more elaborate protocols and security properties. On the other hand, they do not provide the strong computational guarantees that can be obtained through the use of computational models.

Conclusion

As promised in the title of this thesis, we discussed of the modelling and of the analysis of Authenticated Group Key Agreement Protocols.

Guided by case-studies found in the Cliques protocols [6, 7], we first proposed a simple model for the analysis of a particular family of AGKAP. This choice was motivated by several interesting characteristics the Cliques protocols present.

At first, they are built from a very limited set of primitives: modular exponentiation is the only operation performed by principals executing these protocols. However, up to now, this operation was out of the scope of the published logical models (an exception can however be found in the work of Meadows [50] which was a contemporary of ours).

Furthermore, the numerous open-ended protocols (i.e. protocols whose structure may include an arbitrarily large number of data fields) recently published make the analysis of such structures an emerging problem in the field of the analysis of security protocols [51, 52, 67].

Our analysis of the Cliques AGKAP allowed us to discover attacks against the different security properties claimed for the classical A-GDH.2 protocol suite and for the SA-GDH.2 protocol. The development of our model presented another benefit: it allowed us to put into light several flavors of security properties that do not appear when group protocols are considered as simple extensions of two-party protocols.

Discovering attacks against these AGKAP's naturally raised a new problem: can we design a protocol resisting to the attacks captured by our model without changing the design rules adopted by the authors of the Cliques protocols?

We examined this question in Chapter 2. A first step in our study was the definition of a class of protocols containing all those we considered as fix candidates. So, instead of modelling selected aspects of existing protocols as we did in Chapter 1, we needed to be able to specify all (high-level) characteristics of protocol executions, including message routing, which were kept out of our previous model. The cost of this requirement is a heavier formalism, but its benefit is a general result showing that we are able to systematically undermine the key authentication property for all protocols of the family we considered. We think

that this result is quite unusual: as far as we know, the only similar developments are those concerning the security of the ping-pong protocols published from 1982 to 1985 [24, 25, 28, 29]. It would be interesting to investigate in which measure our approach could be adapted to other protocol families.

The results of our first two chapters being not really encouraging from a practical point of view, we turned in Chapter 3 to the design of an AGKAP based on different design assumptions and cryptographic primitives. Starting from the same basic structure as Ateniese & al. [7] and Bresson & al. [13, 14, 17], we built a new AGKAP, the AT-GDH protocol, based on the authentication test design methodology [33].

The close relation between our protocol and the AKE1 proposed in [13] allowed us to illustrate the main differences between these two models assumptions and scope.

Observations all along this thesis are suggesting a number of directions for future research.

Firstly, the AT-GDH protocol is only defined for static groups: once a group of users is sharing a key, the only way to add a member to this group is to restart the whole protocol from the beginning, which is not really efficient. Defining protocols allowing to merge and split groups would therefore be an interesting point. It would also be interesting to examine how other group key constructions could be managed through our method.

More generally, it would also be profitable to develop analysis methods for protocols combining modular exponentiation based primitives and encryption or signature schemes. A first work in this direction have been proposed by Millen and Shmatikov in [55]. This class of protocols could provide interesting case-studies for the generalization of our impossibility proof of Chapter 2.

Furthermore, the logical models we adopted (and the logical models in general) are suffering of the same weakness: we are modelling a quite computationally limited intruder by comparison to the one considered in the classical computational models. For instance, in our first two chapters, we assumed that the only useful computation for the intruder is the modular exponentiation. Computational models adopt an opposite point of view: they state that the intruder can execute any probabilistic polynomial time algorithm and assume the intractability of a few selected problems (like the DDH one). We are defining what the intruder can compute, while computational models define what he cannot compute.

In this direction, it would be interesting to examine in which measure works such as those proposed in [4, 34, 37, 38] could be adapted

to group Diffie-Hellman based protocols. This would allow obtaining strong computational guarantees at the cost of a systematic (and often automatic) logical analysis.

Nowadays, a number of researchers in the cryptography and security community still consider that developing models and proofs of security protocols is an irrelevant activity. The main motivation for this position is that security proofs always rely on a particular model, and that the adversary has no reason to behave as expected in this model.

If a certain amount of scepticism is always appropriate, we think however that security proofs present a number of benefits. Designing models and writing proofs in these models allowed researchers to better understand the role of the different components of the analyzed protocols. This resulted in discovering attacks in many well-known protocols and realizing that many notions as simple as the confidentiality of a message can be understood in many different ways. Putting such notions into light allows designers to specify the intended use of their protocols much more precisely and prevent the final user from misusing them.

Furthermore, security proofs guarantee that an adversary will not be able to undermine security properties by adopting some of the modelled behaviors. Even though this is no panacea, it is always better than no information at all. This last remark encourages us to keep in mind that establishing security proofs must remain an ongoing work.

Bibliography

[1] Federal Information Processing Standards Publication 81. *DES modes of operation*. U.S. Departement of Commerce / National Bureau of Standards, National Technical Information Service, Springfield, Virginia, US, 1980.

[2] M. Abadi. Two facets of authentication. In *11th IEEE Computer Security Foundations Workshop — CSFW'98*, pages 27–32, Rockport, MA, 1998. IEEE Computer Society Press.

[3] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148:1–70, 1999.

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Proceedings of the IFIP International Conference on Theoretical Computer Science 2000*, pages 3–22, Sendai, Japan, 2000. Springer-Verlag - LNCS Vol. 1872.

[5] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. Technical Report CNDS 2001-5, Johns Hopkins University, Center of Networking and Distributed Systems, Nov. 2001. `http://www.cnds.jhu.edu/pub/papers/cnds-2001-5.ps`.

[6] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 17–26, San Francisco, USA, 1998. ACM Press.

[7] G. Ateniese, M. Steiner, and G. Tsudik. New multi-party authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communication*, 18(4):628–639, 2000.

[8] G. Bella, F. Massaci, and L.C. Paulson. The verification of an industrial payment protocol: The SET purchase phase. In V. Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 12–20, Washington DC, USA, 2002. ACM Press.

[9] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of Advances in Cryptology: Crypto'93*, pages 232–249, Santa Barbara, USA, 1994. Springer-Verlag - LNCS Vol. 773.

[10] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Cryptography and Coding*, pages 30–45, Cirencester, UK, 1997. Springer-Verlag - LNCS Vol. 1355.

[11] C. Boyd. Towards extensional goals in authentication protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, Rutgers, USA, 1997.

[12] S. H. Brackin. Using checkable types in automatic protocol analysis. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 99–108, Phoenix, USA, 1999. IEEE Computer Society Press.

[13] E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange - the dynamic case. In C. Boyd, editor, *Advances*

in Cryptology - Proceedings of AsiaCrypt 2001, pages 290–309, Gold Coast, Australia, 2001. Springer-Verlag - LNCS Vol. 2248.

[14] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In L. Knudsen, editor, *Advances in Cryptology - Proceedings of Eurocrypt 2002*, pages 321–336, Amsterdam, the Netherlands, 2002. Springer-Verlag - LNCS Vol. 2332.

[15] E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. In Y. Zheng, editor, *Advances in Cryptology - Proceedings of AsiaCrypt 2002*, pages 497–514, Queenstown, New Zealand, 2002. Springer-Verlag - LNCS Vol. 2501.

[16] E. Bresson, O. Chevassut, and D. Pointcheval. The group Diffie-Hellman problems. In Y. Zheng, editor, *Proceedings of the 9-th Annual Workshop on Selected Areas in Cryptography (SAC'02)*, pages 325–338, St. John's, Canada, 2002. Springer-Verlag - LNCS Vol. 2595.

[17] E. Bresson, O. Chevassut, D. Pointcheval, and J-J Quisquater. Provably authenticated group Diffie-Hellman key exchange. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 255–264, Philadelphia, USA, 2001. ACM Press.

[18] J. Bryans and S. Schneider. CSP, PVS, and a recursive authentication protocol. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, Rutgers, USA, 1997.

[19] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In A. De Santis, editor, *Proceedings of Eurocrypt'94*, pages 275–286, Perugia, Italy, 1994. Springer-Verlag - LNCS Vol. 950.

[20] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[21] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer ans System Sciences*, 18(2):143–154, 1979.

[22] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[23] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[24] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols (extended abstract). In David Chaum, Ronald L. Rivest, , and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of Crypto '82*, pages 177–186, New York, USA, 1982. Plenum Publishing.

[25] D. Dolev and A. C. Yao. On the security of public-key protocols. *IEEE Transactions on information theory*, 2(29):198–208, 1983.

[26] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999. `http://www.mcs.le.ac.uk/~glowe/Security/Papers/prots.ps`.

[27] A. Durante, R. Focardi, and R. Gorrieri. CVS: A tool for the analysis of cryptographic protocols. In *Proceedings of the 12-th IEEE Computer Security Foundations Workshop*, pages 203–212, Mordano, Italy, 1999. IEEE Computer Society Press.

[28] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols (abstract). In David Chaum, Ronald L. Rivest, , and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of Crypto '82*, pages 315–316, New York, USA, 1982. Plenum Publishing.

[29] S. Even, O. Goldreich, and A. Shamir. On the security of ping-pong protocols when implemented using the rsa. In H.C. Williams, editor, *Advances in*

Cryptology: Proceedings of Crypto'85, pages 58–72, Santa Barbara, USA, 1986. Springer-Verlag - LNCS Vol. 218.

[30] R. Focardi and R. Gorrieri. Classification of security properties. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, pages 331–396. Springer-Verlag - LNCS Vol. 2171, 2001.

[31] R. Focardi and R. Gorrieri, editors. *Foundations of Security Analysis and Design*. Springer-Verlag - LNCS Vol. 2171, 2001.

[32] J. Guttman. Security-goals: Packet trajectories and stand spaces. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, pages 197–261. Springer-Verlag - LNCS Vol. 2171, 2001.

[33] J. Guttman. Security protocol design via authentication tests. In *Proceedings of 15th IEEE Computer Security Foundations Workshop*, pages 92–103. IEEE Computer Society Press, 2002.

[34] J. Guttman, F. J. Thayer Fábrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 186 – 195, Philadelphia, USA, 2001. ACM Press.

[35] J. Heather. *Oh! Is it really you? - Using rank functions to verify authentication protocols*. PhD thesis, Royal Holloway, University of London, 2000.

[36] J. Heather and S. Schneider. Towards automatic verification of authentication protocols on an unbounded network. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 132–143, Cambridge, UK, 2000. IEEE Computer Society Press.

[37] J. Herzog. Computational soundness of formal adversaries. Master's thesis, MIT, 2002.

[38] J. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop — CSFW'03*, Asilomar, USA, 2003. IEEE Computer Society Press.

[39] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology - Proceedings of AsiaCrypt'96*, pages 36–49, Kyongju, South Korea, 1996. Springer-Verlag - LNCS Vol. 1163.

[40] J. Katz and M. Yung. Authenticated group key exchange in constant rounds. In *Proceedings of Crypto'03*, Santa Barbara, USA, 2003. Springer-Verlag - LNCS (to appear).

[41] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Proceedings of IFIP-SEC 2001*, pages 229–244, Paris, France, 2001. Kluwer Publishers.

[42] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. *Cryptology ePrint Archive*, 2002/009, 2002. `http://eprint.iacr.org/2002/009.ps`.

[43] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proceedings of Crypto'96*, pages 104–113, Santa Barbara, USA, 1996. Springer-Verlag - LNCS Vol. 1109.

[44] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of Crypto'99*, pages 388–397, Santa Barbara, USA, 1999. Springer-Verlag - LNCS Vol. 1666.

[45] G. Lowe. Some new attacks upon security protocols. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society Press, 1996.

[46] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 31–44, Rockport, USA, 1997. IEEE Computer Society Press.

[47] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[48] W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, Rutgers, USA, 1997.

[49] C. Meadows. The NRL protocol analyzer : an overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[50] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In *Proceedings of the Workshop on Issues in the Theory of Security*, pages 1–4, Geneva, Switzerland, 2000.

[51] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of the DARPA Information Survivability Conference and Exposition — DISCEX 2000*, pages 237–250, Hilton Head, USA, 2000. IEEE Computer Society Press.

[52] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on selected areas in communications*, 21(1), 2003. To appear.

[53] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, July 1999.

[54] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. In *Proceedings of the Workshop on Issues in the Theory of Security*, Portland, USA, 2002. `http://www.dsi.unive.it/IFIPWG1_7/WITS2002/prog/warinschi.ps.gz`.

[55] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop — CSFW'03 (to appear)*, Asilomar, USA, 2003. IEEE Computer Society Press.

[56] J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). In *Proceedings of the 17-th Annual Conference on the Mathematical Foundations of Programming Semantics*, volume 45, Arhus, Denmark, 2001. Electronic Notes in Theoretical Computer Science - Elsevier Science Publishers.

[57] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proceedings of the 7-th USENIX Security Symposium*, pages 201–216, San Antonio, USA, 1998.

[58] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 141–153, Oakland, USA, 1997. IEEE Computer Society Press.

[59] R. Needham and M. Schroeder. Using encryption in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[60] L. C. Paulson. Mechanised proofs for a recursive authentication protocol. In *Proceedings of the 10-th IEEE Computer Security Foundations Workshop*, pages 84–95, Rockport, USA, 1997. IEEE Computer Society Press.

[61] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[62] O. Pereira and J.-J. Quisquater. On the perfect encryption assumption. In *Proceedings of the Workshop on Issues in the Theory of Security*, pages 42–45, Geneva, Switzerland, 2000.

[63] O. Pereira and J-J. Quisquater. A security analysis of the cliques protocols suites. In *Proceedings of the 14-th IEEE Computer Security Foundations Workshop*, pages 73–81, Cap Breton, Canada, 2001. IEEE Computer Society Press.

[64] A. Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, pages 192–202, Hong-Kong, 1999. City University of Hong-Kong Press.

[65] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. In *Electronic Notes in Theoretical Computer Science*, volume 32. Elsevier Science Publishers, 2000.

[66] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[67] P. Ryan and S.A. Schneider. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.

[68] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, to appear, 2003. `http://theory.stanford.edu/people/jcm/papers/tcs-contract-sign.ps`.

[69] V. Shoup. On formal models for secure key exchange - version 4. Technical Report RZ3120, IBM Zurich Research Lab, Nov. 1999.

[70] D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1,2):47–74, 2001.

[71] D. G. Steer, L. Strawczynski, W. Diffie, and M. J. Wiener. A secure audio teleconference system. In *Advances in Cryptology: Proceedings of Crypto'88*, pages 520–528, Santa Barbara, USA, 1988. Springer-Verlag - LNCS Vol. 403.

[72] M. Steiner. *Secure group Key Agreement*. PhD thesis, Universitat des Saarlandes, 2001.

[73] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 31–37, New Delhi, India, 1996.

[74] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A new approach to group key agreement. In *Proceedings of IEEE ICDCS'97*, pages 380–387, Baltimore, USA, 1997. IEEE Computer Society Press.

[75] S. G. Stubblebine and C. A. Meadows. Formal characterization and automated analysis of known-pair and chosen-text attacks. *IEEE Journal on Selected Areas in Communications*, 18(4):571–581, 2000.

[76] P. Syverson and P. van Oorschot. On unifying some cryptographic protocols logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 14–24, Oakland, USA, 1994. IEEE Computer Society Press.

[77] F. J. Thayer and J. Guttman. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[78] F. J. Thayer, J. H. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[79] W. G. Tzeng. A practical and secure fault-tolerant conference key agreement protocol. In *Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC'00*, pages 1–13, Melbourne, Australia, 2000. Springer-Verlag - LNCS Vol. 1751.

APPENDIX A

Publications list

Journal Papers:

- *Some attacks upon authenticated group key agreement protocols*, with J.-J. Quisquater, to appear in Journal of Computer Security, 2003.
- *On the Wagner-Whitin lot-sizing polyhedron*, with L. A. Wolsey, in Mathematics of Operations Research 26(3):591-600, 2001. Also in CORE Discussion Papers No. 23, 2000.

Publications in refereed international conferences:

- *On the Perfect Encryption Assumption*, with J.-J. Quisquater, in Proceedings of the Workshop on Issues in the Theory of Security (WITS 2000), pp. 42-45. Geneva - Switzerland, 2000.
- *Security Analysis of the Cliques Protocols Suites: 1st Results*, with Jean-Jacques Quisquater, in Proceedings of IFIP Sec'01, pp. 151-166, Kluwer Publishers. Paris - France, 2001.
- *A Security Analysis of the Cliques Protocols Suites*, with Jean-Jacques Quisquater, in Proceedings of the 14-th IEEE Computer Security Foundations Workshop, pp. 73-81, IEEE Computer Society Press. Cap Breton - Canada, 2001.

Invited Talks:

- *On the Perfect Encryption Assumption*, with J.-J. Quisquater, presented at the poster session of the European Symposium on Reasearch in Computer Security - ESORICS 2000, Toulouse - France.
- *Two Formal Views of Authenticated Group Diffie-Hellman Key Exchange*, with E. Bresson, O. Chevassut, D. Pointcheval, and J-J. Quisquater, presented at the DIMACS Workshop on Cryptographic Protocols in Complex Environments, 2002, Rutgers - USA.

APPENDIX B

Building an Attack from a Linear System Solution

We consider the systematic reconstruction of an attack against implicit key authentication from the linear system solved in Section 5.1.2 of Chapter 1.

In that section, we considered a first session of the protocol in which M_1, M_2, M_I and M_3 are the intended participants (M_3 being the group controller), and a second session with the same participants excepted M_I. We denoted by r_i and r'_i the random contribution generated by M_i during these two sessions. So, the sets of interest were:

$$\begin{aligned} \mathsf{S} &= \{r_1, r_2, r_I, r_3K_{13}, r_3K_{23}, r_3K_{I3}, \\ &\quad\ r'_1, r'_2, r'_3K_{13}, r'_3K_{23}\} \\ \mathsf{E}_I &= \{r_I, K_{I3}\} \\ \mathsf{P}_S &= \{r'_1K_{13}^{-1}, r'_2K_{23}^{-1}, r'_3\} \end{aligned}$$

And a solution of the linear system corresponding to these sets for the verification of the secrecy of $r'_2K_{23}^{-1}$ is:

$$r'_2 = 1 \qquad r_3K_{23} = -1 \qquad r_3K_{I3} = 1$$

while all other variables are unused.

So, as expressed in Section 4.4.4 of the same chapter, we have to use the services r'_2 and r_3K_{I3} in the positive direction and the service r_3K_{23} in the negative direction.

In order to check whether an attack can be reconstructed from this solution, we now verify the seven conditions defined in Section 4.4.4 of Chapter 1.

Condition 1 When constructing a pair of elements of $\mathcal{G}$, we may exploit at most two services within one single round (and each of them only once), and these services must be exploited in opposite directions.

This condition is verified: r'_2 is the only exploited service during its round, and r_3K_{I3} and r_3K_{23} are exploited in opposite directions.

Condition 2 When constructing a pair of elements of $\mathcal{G}$, we may exploit at most one splitting service and at most two starting services, provided

that they are exploited in opposite directions. Furthermore, if we are using a splitting service, we may not use any starting service.

This condition is verified: we do not exploit splitting services nor starting services. The three next conditions are therefore also trivially verified.

Condition 3 If a splitting service provided by M_x is used when constructing a pair of elements of $\mathcal{G}$, then we cannot use any service M_x should provide before it.

Condition 4 If one starting service provided by M_x is used in a certain direction when constructing a pair of elements of $\mathcal{G}$, then we cannot use any service M_x should provide in the same direction before it.

Condition 5 If two starting services provided by M_x and M_y (it is possible that $M_x = M_y$) are used when constructing a pair of elements of $\mathcal{G}$, then we cannot use the services provided by both users before the considered starting services (but we can use those provided by only one of these users, provided that they are used in the direction opposite to the one of the starting service provided by that user)

We now check the sixth condition:

Condition 6 When attacking M_i, we cannot use in the negative direction any service he provides during the round from which he computes his view of the group key, nor after that round.

M_2 computes the group from the reception of the final broadcast, and does not provide any service from this message. Condition 6 is then verified, and it is also the case of the next one for the same reason.

Condition 7 When attacking M_i, we cannot use any service the input of which is the element of $\mathcal{G}$ that M_i uses to compute his view of the group key.

So, we may apply Algorithms 1 and 2 to build an attack. They may be rewritten as below, taking into account that no splitting nor starting services is used.

So, we have to apply the **Collect** procedure, member after member.

If we consider M_1, we may observe that we are not using any service it provides, so the clauses of the two **if** statements of Algorithm 2 are not verified for all rounds executed by M_1.

If we turn to M_2, we observe that we are using in the positive direction the r'_2 service he provides during the first round of the second

Algorithm 1 Provides a pair (g_1, g_2) the ratio of which is equal to a product of services

Intruder isolates all group members: he is intercepting all messages they are sending
$g_1 := \alpha \quad g_2 := \alpha$
if no splitting service nor starting service is used **then**
  Collect the services member by member
end if

Algorithm 2 Collect the services offered by some member M_x into the pair (g_1, g_2)

for $j := 1$ to the number of rounds executed by M_x **do**
  if some service s provided by M_x during the j-th round has to be used in the negative direction **then**
    g_1 is provided (if possible) as input for s and will be updated with its output
  end if
  if some service s' provided by M_x during the j-th round has to be used in the positive direction **then**
    g_2 is provided (if possible) as input for s' and will be updated with its output
  end if
  Dummy values are given for the unaffected inputs waited in the current round
end for

session. So, we will set the input value of that service to the current value of g_2: α. A dummy value (α^x for instance) is provided as the other element of that message since M_2 is waiting for two values in that round. So, M_2 receives $\langle \alpha^x, \alpha \rangle$ as input and tries to send $\langle \alpha^{xr'_2}, \alpha, \alpha^{r'_2} \rangle$ to M_3, but this message is intercepted and the intruder updates g_2 to $\alpha^{r'_2}$.

$$M_I \xrightarrow{\alpha^x, \alpha} M_2 \xrightarrow{\alpha^{xr'_2}, \alpha, \alpha^{r'_2}} M_I$$

This is the only service provided by M_2 that we have to use. Finally, if we turn to M_3, we observe that we need to use the service $r_3 K_{23}$ in the negative direction and the service $r_3 K_{I3}$ in the positive direction. These services are both provided during the last round of the first session of the protocol (that is the only round executed by M_3 during that session). So, instead of receiving $\langle \alpha^{r_2 r_I}, \alpha^{r_1 r_I}, \alpha^{r_1 r_2}, \alpha^{r_1 r_2 r_I} \rangle$ as given in the protocol definition, the intruder sets the second element of this sequence to g_1 and the third to g_2 while placing dummy values in the two remaining positions (say α^y and α^z). The message M_3

receives will then be $\langle \alpha^y, \alpha, \alpha^{r'_2}, \alpha^z \rangle$, and he will therefore broadcast $\langle \alpha^{yr_3K_{13}}, \alpha^{r_3K_{23}}, \alpha^{r'_2r_3K_{I3}} \rangle$.

$$M_I \xrightarrow{\alpha^y,\alpha,\alpha^{r'_2},\alpha^z} M_3 \xrightarrow{\alpha^{yr_3K_{13}},\alpha^{r_3K_{23}},\alpha^{r'_2r_3K_{I3}}} M_I$$

The intruder then updates g_1 to $\alpha^{r_3K_{23}}$ and g_2 to $\alpha^{r'_2r_3K_{I3}}$.

We have now used all necessary services, and the pair $(g_1, g_2) = (\alpha^{r_3K_{23}}, \alpha^{r'_2r_3K_{I3}})$ has the form expected (except concerning the presence of K_{I3} that the intruder can easily suppress).

At that time, M_2 is still waiting for the final broadcast of the second session he started and during which his contribution to the group key was r'_2. So, the intruder will send him this broadcast, placing $\alpha^{r_3K_{23}}$ in second position so that M_2 will compute its view of the group key from it. The first element of this message may be set to any value.

$$M_I \xrightarrow{\alpha^x,\alpha^{r_3K_{23}}} M_2$$

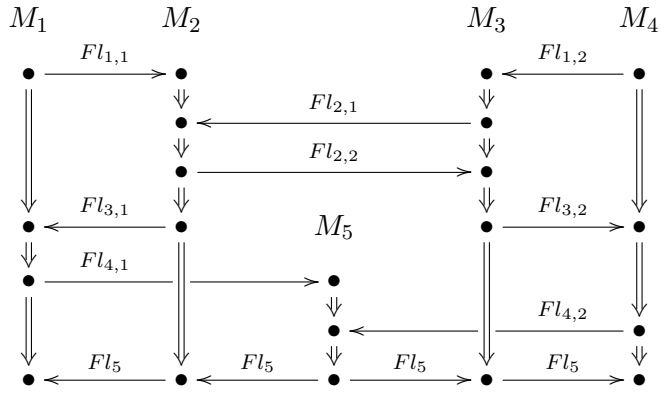
M_2 will therefore compute $\alpha^{(r_3K_{23})r'_2K_{23}^{-1}} = \alpha^{r'_2r_3}$ as group key, value that the intruder can easily compute from g_2.

APPENDIX C

Illustration of Chapter 2's Attack Process

We illustrate the full attack construction process developed all along this thesis second chapter. To this purpose, we define a deliberately intricate protocol, the Int-GDH protocol, which will allow us to illustrate our attack construction process more completely than if we considered a simple, regular protocol.

A typical execution of the Int-GDH protocol is represented in the strand space of Fig. C.1.



$$
\begin{aligned}
Fl_{1,1} &= \alpha^{r_1}, \alpha^{r_1 K_{15}} \\
Fl_{1,2} &= \alpha^{r_4} \\
Fl_{2,1} &= \alpha^{r_4}, \alpha^{r_3} \\
Fl_{2,2} &= \alpha^{r_2 K_{12} K_{25}}, \alpha^{r_1}, \alpha^{r_1 r_2 K_{15} K_{25}} \\
Fl_{3,1} &= \alpha^{r_2 r_4}, \alpha^{r_2 r_3} \\
Fl_{3,2} &= \alpha^{r_2 r_3 K_{12} K_{25}}, \alpha^{r_1 r_3}, \alpha^{r_1 r_2 r_3 K_{15} K_{25} K_{35}} \\
Fl_{4,1} &= \alpha^{r_1 r_2 r_4}, \alpha^{r_1 r_2 r_3} \\
Fl_{4,2} &= \alpha^{r_2 r_3 r_4 K_{12} K_{25}}, \alpha^{r_1 r_3 r_4}, \alpha^{r_1 r_2 r_3 r_4 K_{15} K_{25} K_{35} K_{45}} \\
Fl_5 &= \alpha^{r_2 r_3 r_4 r_5 K_{15}}, \alpha^{r_1 r_3 r_4 r_5 K_{25}}, \alpha^{r_1 r_2 r_4 r_5 K_{35}}, \alpha^{r_1 r_2 r_3 r_5 K_{45}}
\end{aligned}
$$

FIGURE C.1. A run of the Int-GDH protocol

Even though they can be easily deduced from the strand definitions, we give the five histories in Table C.1.

TABLE C.1. Histories in the Int-GDH Protocol

α_1	α_2	α_3	α_4	α_5
$(\langle s_2, 3\rangle, 1)$	$(\langle s_1, 1\rangle, 1)$	$(\langle s_4, 1\rangle, 1)$	$(\langle s_3, 2\rangle, 2)$	$(\langle s_1, 1\rangle, 2)$
$(\langle s_3, 3\rangle, 1)$	$(\langle s_2, 1\rangle, 2)$	$(\langle s_3, 1\rangle, 1)$	$(\langle s_2, 2\rangle, 2)$	$(\langle s_2, 1\rangle, 2)$
$(\langle s_3, 4\rangle, 1)$	$(\langle s_2, 3\rangle, 2)$	$(\langle s_3, 2\rangle, 1)$	$(\langle s_2, 4\rangle, 2)$	$(\langle s_2, 3\rangle, 3)$
$(\langle s_4, 2\rangle, 1)$	$(\langle s_3, 3\rangle, 2)$	$(\langle s_2, 2\rangle, 1)$	$(\langle s_1, 2\rangle, 2)$	$(\langle s_3, 3\rangle, 3)$
$(\langle s_4, 3\rangle, 1)$	$(\langle s_3, 4\rangle, 2)$	$(\langle s_2, 4\rangle, 1)$	$(\langle s_1, 3\rangle, 2)$	$(\langle s_3, 4\rangle, 3)$
$(\langle s_5, 2\rangle, 1)$	$(\langle s_4, 2\rangle, 2)$	$(\langle s_1, 2\rangle, 1)$	$(\langle s_5, 1\rangle, 2)$	$(\langle s_4, 2\rangle, 3)$
$(\langle s_5, 3\rangle, 1)$	$(\langle s_4, 3\rangle, 2)$	$(\langle s_1, 3\rangle, 1)$	$(\langle s_5, 3\rangle, 4)$	$(\langle s_4, 3\rangle, 3)$
$(\langle s_1, 4\rangle, 1)$	$(\langle s_5, 2\rangle, 2)$	$(\langle s_5, 1\rangle, 1)$	$(\langle s_4, 4\rangle, 4)$	$(\langle s_5, 2\rangle, 3)$
	$(\langle s_5, 3\rangle, 2)$	$(\langle s_5, 3\rangle, 3)$		
	$(\langle s_2, 5\rangle, 2)$	$(\langle s_3, 5\rangle, 3)$		

We now have a complete definition of the Int-GDH protocol: strands inform us about the way messages are (normally) exchanged, while histories indicate us how they are computed.

We will now build an attack against this protocol.

We first have to select

- three group members: M_i, M_j and M_k
- two disjoint sets of users S_j and S_k such that $M_k \in \mathsf{S}_j$, $M_j \in \mathsf{S}_k$, $M_i \notin \mathsf{S}_j \cup \mathsf{S}_k$, $\mathsf{S}_j \cup \mathsf{S}_k \cup \{M_i\} = \mathsf{M}$.

This selection must also respect the two following conditions:

- $split(M_i, M_j) \neq M_i$
- the product $p =$

$$C^{-1}(M_i \to M_i) \cdot C(M_i \to M_j) \cdot$$
$$[\mathsf{S}_j \backslash M_I : C^{-1}(M_i \to M_j) \cdot C(M_i \to M_k)] \cdot$$
$$\prod_{M_l \in \mathsf{S}_k} [\mathsf{S}_j \backslash M_I : C^{-1}(M_l \to M_i) \cdot C(M_l \to M_k)] \cdot$$
$$\prod_{M_l \in \mathsf{S}_j} [\mathsf{S}_k \backslash M_I : C^{-1}(M_l \to M_i) \cdot C(M_l \to M_j)] \cdot \prod_{M_l \in \mathsf{M}} K_{Il}^{e_l}$$

respects at least one of the conditions described in Proposition 2.13, namely:

(1) p contains at most one splitting point and no starting point;
(2) p contains no splitting point, one start$^+$ and no start$^-$;
(3) p contains no splitting point, no start$^+$ and one start$^-$;

(4) p contains no splitting point, one start$^+$ and one start$^-$; both occurring for the index $i \in \mathcal{I}$);
(5) p contains no splitting point, one start$^+$ (for the index $i_+ \in \mathcal{I}$), one start$^-$ (for the index $i_- \in \mathcal{I}$, $i_+ \neq i_-$) and $C(M_{j_{i_-}} \rightarrow M_{k_{i_-}}) \prec C(M_{j_{i_-}} \rightarrow M_{l_{i_-}})$ or $C(M_{j_{i_+}} \rightarrow M_{l_{i_+}}) \prec C(M_{j_{i_+}} \rightarrow M_{k_{i_+}})$.

We first observe that the five histories of the Int-GDH protocol have no common part, so that there are no splitting point.

As a first try, we consider the choice $M_i = M_1$, $M_j = M_2$ and $M_k = M_3$. Whatever choice we do for S_j and S_k, we can verify that the product p will contain at least three starting points: $C(M_1 \rightarrow M_2)$, $[\mathsf{S}_j \backslash M_I : C^{-1}(M_1 \rightarrow M_2)]$ and $[\mathsf{S}_j \backslash M_I : C^{-1}(M_2 \rightarrow M_1)]$. These values of M_i, M_j and M_k are therefore not admissible.

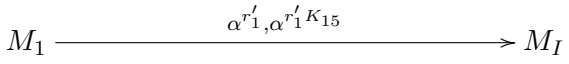
As a second attempt, we consider the choice $M_i = M_1$, $M_j = M_3$, $M_k = M_2$ while $\mathsf{S}_j = \{M_2\}$ and $\mathsf{S}_k = \{M_3, M_4, M_5\}$. This solution implies that p contains one *start*$^+$: $[\mathsf{S}_j \backslash M_I : C(M_1 \rightarrow M_2)]$ and one *start*$^-$: $[\mathsf{S}_k \backslash M_I : C^{-1}(M_2 \rightarrow M_1)]$. As expressed in our fifth condition, this is acceptable only if $C(M_1 \rightarrow M_2) \prec C(M_1 \rightarrow M_3)$ or $C(M_2 \rightarrow M_1) \prec C(M_2 \rightarrow M_3)$. A simple verification in Table C.1 shows that $C(M_2 \rightarrow M_1) \not\prec C(M_2 \rightarrow M_3)$ because $\langle s_2, 2\rangle \prec \langle s_2, 3\rangle$. However, we can verify that $C(M_1 \rightarrow M_2) \prec C(M_1 \rightarrow M_3)$ since $\langle s_1, 1\rangle$ strictly precedes all nodes of α_3 belonging to s_1. We are therefore able to build an attack for this selection of values.

A simple way to construct our attack consist in following the procedure explained in the fifth part of the Proposition 2.13 proof.

The first step in this procedure consists in defining $\hat{z}$ as the index of the starting point of α_2 in s_1 (given that $C(M_1 \rightarrow M_2) \prec C(M_1 \rightarrow M_3)$). A simple examination shows that $\hat{z} = 1$.

We now have to execute Algorithm 4 for the product $[M_2 \backslash M_I : C^{-1}(M_1 \rightarrow M_3) \cdot C(M_1 \rightarrow M_2)]$ and for values of z ranging from 1 to $\hat{z}$, what means that we will execute this algorithm for only one step of the **for** loop. The values g_1 and g_2 are initialized to α and, for the simplicity of the writings, we always select α^x as random element of G. The random contribution of M_i during the session we are attacking will be written r_i, while his contribution during the session where the intruder replaces the users included in S_j will be denoted r_i', and we will use the letter r_i'' to write the contribution M_i generated during the session where the intruder replaces the users included in S_k. The strand space resulting from this partial execution of Algorithm 4 is represented in Fig. C.2. The current values of g_1 and g_2 are indicated as well.

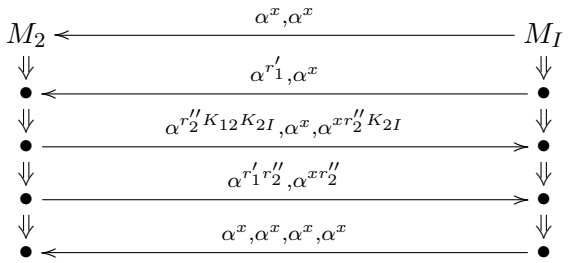
Always following the procedure indicated in the fifth part of the proof of Proposition 2.13, we now have to execute Algorithm 4 for the



$$g_1 = \alpha,\ g_2 = \alpha^{r'_1}$$

FIGURE C.2. First Step

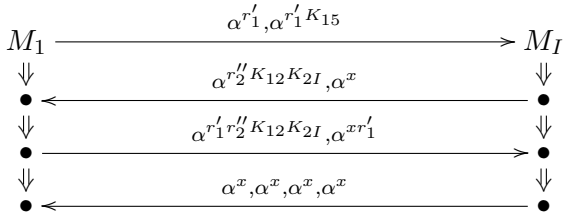
product $[\{M_3, M_4, M_5\} \backslash M_I : C^{-1}(M_2 \rightarrow M_1) \cdot C(M_2 \rightarrow M_3)]$, keeping the current values of g_1 and g_2 as initial values. The resulting strand space is represented in Fig. C.3.



$$g_1 = \alpha^{r''_2 K_{12} K_{2I}},\ g_2 = \alpha^{r'_1 r''_2}$$

FIGURE C.3. Second Step

The next step in the procedure described in the fifth part of the proof of Proposition 2.13 consists in completing the execution of Algorithm 4 for the product $[M_2 \backslash M_I : C^{-1}(M_1 \rightarrow M_3) \cdot C(M_1 \rightarrow M_2)]$. The result of this execution (with the updated values of g_1 and g_2) is represented in Fig. C.4.



$$g_1 = \alpha^{r'_1 r''_2 K_{12} K_{2I}},\ g_2 = \alpha^{r'_1 r''_2}$$

FIGURE C.4. Third Step

We now have to execute Algorithm 4 for the remaining products of pairs of contributions in

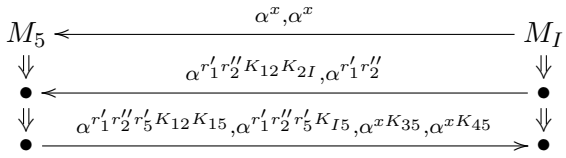
$$\begin{aligned} p \;=\; & C^{-1}(M_1 \to M_1) \cdot C(M_1 \to M_3) \cdot \\ & [M_2 \backslash M_I : C^{-1}(M_1 \to M_3) \cdot C(M_1 \to M_2)] \cdot \\ & \prod_{M_l \in \{M_3, M_4, M_5\}} [M_2 \backslash M_I : C^{-1}(M_l \to M_1) \cdot C(M_l \to M_2)] \cdot \\ & [\{M_3, M_4, M_5\} \backslash M_I : C^{-1}(M_2 \to M_1) \cdot C(M_2 \to M_3)] \cdot \prod_{M_l \in \mathsf{M}} K_{Il}^{e_l} \end{aligned}$$

A direct observation however shows that

$$[M_2 \backslash M_I : C^{-1}(M_3 \to M_1) \cdot C(M_3 \to M_2)] = 1$$
$$[M_2 \backslash M_I : C^{-1}(M_4 \to M_1) \cdot C(M_4 \to M_2)] = 1$$

so we do not need to apply Algorithm 4 for these products in our protocol.

We however have to collect the products $[M_2 \backslash M_I : C^{-1}(M_5 \to M_1) \cdot C(M_5 \to M_2)]$ and $C^{-1}(M_1 \to M_1) \cdot C(M_1 \to M_3)$ by applying the same process as before. Always keeping the current values of g_1 and g_2, the strand space obtained for the product $[M_2 \backslash M_I : C^{-1}(M_5 \to M_1) \cdot C(M_5 \to M_2)]$ is represented in Fig. C.5.

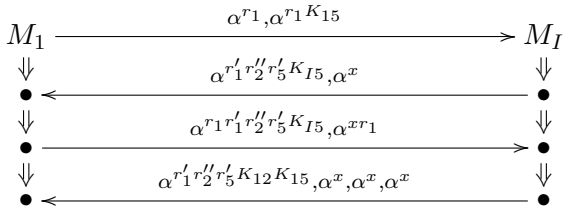


$$g_1 = \alpha^{r_1' r_2'' r_5' K_{12} K_{15}},\ g_2 = \alpha^{r_1' r_2'' r_5' K_{I5}}$$

FIGURE C.5. Fourth Step

In order to complete our attack, we still have to execute Algorithm 4 for the product $C^{-1}(M_1 \to M_1) \cdot C(M_1 \to M_3)$ and to send g_1 as the value M_1 will use to compute the group key. This is represented in Fig. C.6.

At the end of this process, when M_1 will compute his view of the group key, he will exponentiate $g_1 = \alpha^{r_1' r_2'' r_5' K_{12} K_{15}}$ with $r_1 K_{12}^{-1} K_{15}^{-1}$ and obtain $\alpha^{r_1 r_1' r_2'' r_5'} = g_2^{K_{I5}^{-1}}$. The intruder can therefore compute a key that would normally have to be kept secret.



$$g_1 = \alpha^{r'_1 r''_2 r'_5 K_{12} K_{15}},\ g_2 = \alpha^{r_1 r'_1 r''_2 r'_5 K_{I5}}$$

FIGURE C.6. Last Step