IST-2002-507932

ECRYPT

European Network of Excellence in Cryptology

Network of Excellence

Information Society Technologies

D.VAM.3
Hardware Crackers

Due date of deliverable: 31. July 2005
Actual submission date: 25. August 2005

Start date of project: 1. February 2004 Duration: 4 years

Lead contractor: Institute for Applied Information Processing and Communications (IAIK)

Revision 1.0

Project co-funded by the European Commission within the 6th Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	

Hardware Crackers

Editor
Elisabeth Oswald (IAIK)

Contributors
Lejla Batina (KUL), Nele Mentens (KUL), Elisabeth Oswald (IAIK), Jan Pelzl (RUB), Christine Priplata (EDI), Colin Stahlke (EDI) and Francois-Xavier Standaert (UCL)

25. August 2005
Revision 1.0

The work described in this report has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932.

Contents

Chapter 1

Executive Summary

The reason for investigating hardware is the chance to build more efficient (cheaper and faster) crackers than would be possible in software and therefore to obtain a completely different quality of cracking power. The public interest in designing crackers is on the one hand to get aware of the dangers of leaking information and on the other hand to get convincing estimates about the security of widely used algorithms and key lengths. Given the increasing number of applications depending on cryptographic mechanisms, there is a strong need for further research in this area.

This deliverable reviews the state-of-the-art research that was discussed during the SHARCS workshop. The SHARCS workshop, organized by ECRYPT's VAMPIRE Lab in February 2005, was the first open meeting devoted entirely to this challenging subject of hardware crackers. The field of special-purpose hardware for attacking cryptographic systems still has to be defined. Adi Shamir e.g. proposed a conceptual view, banning totally impractical hardware designs and supporting improvements of designs by constant factors. While presenting a cryptanalytic hardware architecture optimized for full cost Michael Wiener discussed the quite general question of finding the right measure to determine the costs of an attack (see also [35] in Chapter 3).

The field of integer factorization with the help of special-purpose hardware attracted more and more attention since 1999 (see e.g. [1], [29], [31], [14], [15], [24] and [23] in Chapter 3). This was reflected by a majority of contributions on that topic during the SHARCS workshop and gave a quite comprehensive summary of the state of the art of that specific but productive area. With successful integer factorization one can crack RSA moduli. RSA is used in many cryptographic protocols and in most security products. Therefore it is of permanent interest for the industry, government authorities and for users to get assurance about the actual security of the current RSA key-length. Of particular interest is the SHARK device that was proposed during the workshop. The SHARK device follows an approach that allows for a practical implementation on state-of-the-art technology. It has a modular architecture of small ASICs connected mainly via standard buses. It consists of 2300 identical isolated machines sieving in parallel. Each machine costs 70 000 US$. SHARK performs the sieving step of GNFS for a 1024-bit number within a year and costs around 200 million US$ (this includes 40 million US$ for the power consumption for one year).

Most contributions of the SHARCS workshop were devoted to asymmetric cryptography. However, DES crackers and DES-based password crackers received attention as well. During SHARCS it was shown how to crack Unix passwords using FPGA platforms (see [8] in Chap-

ter 4) and Jean-Jacques Quisquater gave a report on the costs for cracking DES, especially using a time-memory tradeoff attack (see [20] in Chapter 4). All of the state-of-the-art attacks can be performed with standard equipment such as FPGAs and PCs. As a practical estimation, a US$ 12 000 machine could break DES in about 3 days, presently or in the near future.

Chapter 2

Introduction

Being able to ensure the security (confidentiality, authenticity or integrity) of information has become one of the most important issues in the modern information society. In order to provide certain security properties, cryptographic algorithms have been designed and are used in many (security sensitive) applications, such as banking, GSM, etc., all over the world. The security of the different cryptographic algorithms is therefore very important for all practical applications using them.

There are several established notions of the security of an algorithm. They all assess the security of an algorithm in a specific security model. Practical security models are computational security and ad-hoc security; the confidence in the amount of security of a cryptographic algorithm based on computational or ad-hoc security increases with time and investigation of the algorithm. Of course, time alone is not enough if only very few people have tried to break the primitive.

In the last years, the interest in the practical security of algorithms has started to grow again. Researchers have started to investigate the possibility of designing special purpose software and hardware to assess the practical security of algorithms.

The motivation for designing and building hardware crackers are the same as for software crackers: supporting the cracking of cryptographically protected systems by doing calculations on the data available to the attacker. These data can be openly available like an encrypted message, a public key, a signature, or they can be part of secret information obtained by more sophisticated means like side channels or deception.

The reason for investigating hardware is the chance to build more efficient (cheaper and faster) crackers than would be possible in software and therefore to obtain a completely different quality of cracking power. The public interest in designing crackers is on the one hand to get aware of the dangers of leaking information and on the other hand to get convincing estimates about the security of widely used algorithms and key lengths. Given the increasing number of applications depending on cryptographic mechanisms, there is a strong need for further research in this area.

2.1 The SHARCS Workshop

The Special-purpose Hardware for Attacking Cryptographic Systems (SHARCS) workshop, organized by ECRYPT's VAMPIRE Lab in February 2005, was the first open meeting devoted entirely to this challenging subject of hardware crackers. The field of special-purpose hard-

ware for attacking cryptographic systems still has to be defined. Adi Shamir e.g. proposed a conceptual view, banning totally impractical hardware designs and supporting improvements of designs by constant factors. While presenting a cryptanalytic hardware architecture optimized for full cost Michael Wiener discussed the quite general question of finding the right measure to determine the costs of an attack (see also Chapter 3, [35] in Chapter 3).

The SHARCS workshop concentrated mainly on two types of hardware crackers. The major number of contributions was dedicated to devices for integer factorization. The other contributions focused on devices for DES cracking (or DES-based password cracking).

The field of integer factorization with the help of special-purpose hardware attracted more and more attention since 1999 (see e.g. [1], [29], [31], [14], [15], [24], [23] in Chapter 3). This was reflected by a majority of contributions on that topic during the SHARCS Workshop and gave a quite comprehensive summary of the state of the art of that specific but productive area. With successful integer factorization one can crack RSA moduli. RSA is used in many cryptographic protocols and in most security products. Therefore it is of permanent interest for the industry, government authorities and for users to get assurance about the actual security of the current RSA key-length.

The best known algorithm for integer factorization is the General Number Field Sieve (GNFS). However, other number theoretic computations can also lead to breaking the security of RSA. GNFS consists of four main steps: choice of polynomials, sieving to collect relations, a linear algebra step dealing with a huge sparse matrix over the field $\mathbb{F}_2$ and the final computation of the root. The two expensive parts are the sieving and the matrix step. Hence they are the major targets for an implementation in dedicated hardware. There are several choices for the implementation of GNFS in terms of algorithms, parameters, optimization tricks and other details. The choice of parameters changes a lot when one switches between crackers for different bit lengths (in software as well as in hardware). Often, the whole strategy has to be adapted.

The recently proposed devices for integer factorization before SHARCS were TWINKLE (1999, Shamir, see [29] in Chapter 3), Factoring Circuits (2001, Bernstein, see [1] in Chapter 3), TWIRL (2003, Shamir and Tromer, see [31] in Chapter 3) and YASD (2004, Geiselmann and Steinwandt, see [15] in Chapter 3). TWINKLE is a sieving device based on optoelectronics to collect the relations using the Quadratic Sieve or the Number Field Sieve, at the time proposed to break 512-bit RSA moduli very quickly. Bernsteins Factoring Circuits actually address both the sieving and the matrix step of NFS by suggesting a highly parallel approach of a sorting network to distribute the work efficiently. These ideas were developed further by Geiselmann and Steinwandt (see also [14] in Chapter 3) leading e.g. to YASD (Yet Another Sieving Device) and also inspiring TWIRL. The TWIRL device is based on parallel processing pipelines and is designed for 1024-bit integers. Concerns about the feasibility of such giant monolytic ASIC architectures have been raised.

During SHARCS another sieving architecture for 1024-bit integers called SHARK (2004; Franke, Kleinjung, Paar, Pelzl, Priplata, Stahlke, see [12] in Chapter 3) was suggested. SHARK is a parallelized lattice sieving device, which has a modular architecture and consists of small ASICs connected by a specialized butterfly transport system. Note that the other proposals use line sieving instead of lattice sieving. Because of the modular architecture based on small ASICs it is claimed that the SHARK device can be build with today's technology. SHARK as presented in the paper can also be considered as a strategy to address large sieving problems. There are many reasonable modifications and one can play with the parameters. ECM could be used more intensely to permit less sieving. For higher bit lengths the role of

an efficient hardware to factorize the cofactors (like using ECM in ASICs) becomes more and more important. A more profound study of that ECM support with actual implementation results on an FPGA was carried out in parallel during the development of SHARK (see [13] in Chapter 3). SHARK relies on a new approach to lattice sieving by Franke and Kleinjung (see [10] in Chapter 3), covering many SHARK-like designs. This kind of lattice sieving would also save costs for TWIRL. In addition, a SHARK-like machine with certain modifications and appropriate parameters could be used for attacking the DL problem in $\mathbb{F}_p^*$.

Reports on the matrix step of NFS were given by Adi Shamir and Rainer Steinwandt. In particular a systolic design for supporting the matrix-by-vector multiplication as occurring in Wiedemann's algorithm was presented (see [32] and [18] in Chapter 3).

Dan Bernstein pointed out that hardware crackers might become less important if the architecture of general purpose computers changes from a large processor with a lot of memory to a mesh of thousands or millions of small processors, each having its own memory. Such a computer would not only be more suitable for cracking cryptographic systems, it would also fit many other challenges of the modern world, like e.g. pattern recognition. It remains the question how to program such computers, the knowledge about specialized hardware might help there.

Still there is a large gap between theory and practice. Takeshi Shimoyama illustrated that realizing specialized hardware needs enormous development costs and the results are often disappointing. Nevertheless, in real world applications of cryptography there are many more starting points for a possible attack than the cryptographic primitives. Antony E. Sale gave an example of a hardware cracker used in World War II to crack an unknown cipher using real world intercepted data, exploiting design flaws and negligent use of the cipher.

In general, the field of hardware crackers will profit a lot from the cooperation of researchers with quite different backgrounds. The development takes place on the algorithmic, software and hardware level. Therefore the continuous exchange between mathematicians, computer scientists and engineers is crucial for success. It will also lead to a better definition of the scope of the field of special-purpose cryptanalytical machines.

Most contributions of the SHARCS Workshop were devoted to asymmetric cryptography. However, DES crackers and DES-based password crackers received attention as well. During SHARCS it was shown how to crack Unix passwords using FPGA platforms (see [8] in Chapter 4) and Jean-Jacques Quisquater gave a report on the costs for cracking DES, especially using a time-memory tradeoff attack (see [20] in Chapter 4).

2.2 Organization of this Deliverable

The remainder of this deliverable is organized as follows.

In Chapter 3 we report on devices to assist integer factorization. We discuss the GNFS in Section 3.2, and sketch the TWINKLE and TWIRL devices in Section 3.4. The concept of the SHARK device, which is being developed in part by ECRYPT members, is discussed in Section 3.5.

In Chapter 4, we focus on DES cracking devices. The history of DES crackers is reviewed in Section 4.1. Exhaustive key search using FPGAs is discussed in Section 4.2. Thereafter, we discuss time-memory tradeoffs and their implementation in practice in Section 4.3. These practical implementations have been developed in part by ECRYPT members.

We conclude this report in Chapter 5.

Chapter 3

Factoring Hardware

3.1 Factoring Integers

Factoring large integers is a classical mathematical problem which became a rich and complex research area within algorithmic number theory. With the introduction of asymmetric cryptography in the 1970's and the use of RSA in most cryptographic applications and products, the field became an important backbone of the whole world of information security. Factoring the RSA modulus is up to now the only known way the break general RSA.

Today RSA is mostly used with 1024 bit. In order to give estimates about the security of such cryptographic systems, it is important to know the time and the cost for factoring the RSA modulus. Looking at complexity and asymptotic estimates can only lead to guesses about the security. Also looking at MIPS years needed for factoring is not enough, since the total cost for a MIPS year heavily depends on the hardware. The only way to get a reliable estimate is to build a machine and to look at the total cost in dollars and seconds it takes to factor an RSA modulus. Since this approach is too costly, the way to go is to design special hardware which is as cheap as possible, and to estimate the cost for a factorization in dollars and seconds.

The best known algorithm to factor RSA moduli is the General Number Field Sieve (GNFS). It consists of five main steps: choice of polynomials, sieving to collect relations, generating a huge sparse matrix over the field $\mathbb{F}_2$, a linear algebra step to solve this matrix, and the final computation of the roots. The two expensive parts are the sieving and the matrix step. These two steps should be realized in special-purpose hardware, for the other steps general-purpose computers are sufficient.

During the sieving step many medium sized integers (e.g. between 80 and 200 bit, depending on the choice of parameters) need to be factored. For this "cofactorization" one needs other factoring algorithms. Besides trial division, Pollard-ρ and Pollard-$(p-1)$ one uses MPQS (Multiple Polynomial Quadratic Sieve) in software for the cofactorization. In hardware ECM (Elliptic Curve Method) seems to be the best choice, since it does not need much memory and is easily parallelizable.

3.2 The General Number Field Sieve

Let N be the integer to be factored. The General Number Field Sieve (GNFS) consists of the following steps:

1. Choice of polynomials: Searching for appropriate polynomials $f_0, f_1 \in \mathbb{Z}[x]$
2. Sieving step: Searching for enough pairs $(a, b) \in \mathbb{Z}^2$
3. Generating the matrix
4. Matrix step: Solving the matrix
5. Square root step: Examining the solutions to find the factorization

The aim of GNFS is the same as for the quadratic sieve or for MPQS: find enough numbers $\alpha, \beta \in \mathbb{Z}$ such that $\alpha^2 \equiv \beta^2 \pmod{N}$. Then N divides the product $(\alpha + \beta)(\alpha - \beta)$ and the hope is that $\gcd(N, \alpha - \beta)$ is a non trivial factor of N. The solutions of the matrix give rise to such pairs (α, β).

Each column of the matrix corresponds to a pair (a, b) of the sieving step. If enough pairs (a, b) are found then the matrix has some solutions. Around hundred solutions are usually enough to find a pair (α, β) such that $\gcd(N, \alpha - \beta)$ is a non trivial factor of N.

3.2.1 Complexity

An integer N can be factored in subexponential time with respect to its size. More precisely let

$$L_N(\alpha, \beta) := e^{(\beta + o(1))(\log N)^{\alpha}(\log \log N)^{1-\alpha}}.$$

Then for a suitable choice of parameters in GNFS, the complexity is heuristically

$$L_N\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right).$$

Before the invention of GNFS the best algorithm for factoring large numbers was MPQS. Still in the 1990's it was not clear that the better asymptotics of GNFS would mean that GNFS was also better in practice than MPQS. Since several years all factoring records have been achieved with the number field sieve. They all used general-purpose computers and no special hardware. The actual record for factoring RSA moduli is 663 bit (RSA-200, see [11]).

3.2.2 Choice of Polynomials

This step is done with general-purpose computers. It is not clear, if a massive use of special hardware could be beneficial.

The choice of polynomials $f_0, f_1 \in \mathbb{Z}[x]$ is crucial for the size of the matrix and the running time of the entire algorithm. At least a few hours or days of computation time should be used to find good polynomials. The polynomials f_0 and f_1 have the following properties:

- $f_0, f_1 \in \mathbb{Z}[x]$ irreducible, non constant, coprime,
- gcd(coefficients of f_i) $= 1$,
- $\exists\ m \in \mathbb{Z}$ such that $f_0(m) \equiv f_1(m) \equiv 0 \pmod{N}$.

Let $F_0, F_1 \in \mathbb{Z}[x, y]$ be the homogenized polynomials for f_0 and f_1. An integer is called smooth if it has only small factors. The aim is to find polynomials f_0, f_1, such that there are many $(a, b) \in \mathbb{Z}^2$ such that $F_0(a, b)$ and $F_1(a, b)$ are smooth. Finding more such pairs (a, b) results in a matrix which has more columns. Smoother values of the polynomials result in a matrix which has fewer rows. Both effects lead to more solutions of the matrix.

The actual factoring records use polynomials of degree 1 and 5. A better choice would be, e.g., 3 and 3, but up to now nobody knows how to find good polynomials which both have degree larger than 2. Therefore in practice one polynomial is always linear. Each polynomial leads to one sieve. The linear polynomial leads to the "rational sieve" (the corresponding number field is just $\mathbb{Q}$ here), the other polynomial leads to the "algebraic sieve".

A good strategy to find good polynomials is described in [20].

3.2.3 Sieving Step

For large N this step should probably be done with special hardware. The present factoring records used general-purpose computers for this step. It took roughly 80% of the computing time for the whole factoring process (see [9] and [11]).

The aim of the sieving step is to find many coprime pairs $(a, b) \in \mathbb{Z}^2$ with $b > 0$ such that $F_0(a, b)$ and $F_1(a, b)$ are smooth simultaneously. A more complete introduction to the GNFS can be found in [22].

The number $F_i(a, b)$ is B_i-smooth, if all factors are smaller than B_i. The sieving is done with the following two factor bases for $i = 0, 1$:

$$\mathcal{F}_i = \{(p, r) \quad | \quad p < B_i \quad \text{a prime and} \quad r \in \{0, \ldots, p-1, \infty\} \text{ with } f_i(r) \equiv 0 \mod p\},$$

where $f_i(\infty) \equiv 0 \mod p$ means that the leading coefficient of f_i is divisible by p. For coprime a, b we have

$$p \mid F_i(a, b) \Longleftrightarrow a \equiv br \mod p \text{ for some } (p, r) \in \mathcal{F}_i.$$

For $r = \infty$ the condition is $b \equiv 0 \mod p$.

Line Sieving

The aim is to find in the shortest possible time many (a, b) in the rectangle $|a| \leq A$, $0 < b \leq B$, such that $F_0(a, b)$ and $F_1(a, b)$ are smooth. More precisely, we want many coprime pairs (a, b) such that $F_i(a, b) = G_i R_i$, G_i is B_i-smooth and R_i is a smooth cofactor that has to be factorized by another method, like ECM in special hardware.

For every $b = 1, 2, 3, \ldots, B$ an array (a line) of length $2A+1$ is initialized with zeroes. The positions in this array correspond to the numbers $a = -A, \ldots, A$. For every element of the first factor base $\mathcal{F}_0$ the first a has to be found which satisfies $a \equiv br \bmod p$. To this position the value $\log(p)$ is added. Then $\log(p)$ is added to the positions $a + p$, $a + 2p$,... In practice one adds a scaled approximation, roughly proportional to $\log(p)$, maybe an integer that fits in a byte. In the end position a of the array contains a sum of logarithms of primes that divide $F_0(a, b)$. If this sum is large enough (i.e. $F_0(a, b)$ is smooth enough), the pair (a, b) is a survivor of the first sieve. The procedure has to be repeated with the second factor base $\mathcal{F}_1$. Pairs (a, b) that survive both sieves and the factorization of the cofactors are sieving reports that will give a column of the matrix for the next steps.

Lattice Sieving

Let q be a prime number just a little bit bigger than the largest prime of the factor base and let r be a zero of $f_1 \bmod q$. This means that (q, r) would be in $\mathcal{F}_1$ if B_1 was just a little bigger. Such a pair (q, r) is called a "special q". There are also variants where the special q belongs to the factor base.

In the sieving only these pairs (a, b) are considered, for which $a \equiv br \pmod q$. Therefore $F_1(a, b)$ is, a priori, divisible by q, so the probability that $F_1(a, b)$ is smooth is bigger. Such pairs (a, b) form a lattice. For small primes in the factor base sieving is done in this lattice as in the case of line sieving. But at some point the sieving intervals become too short and the cost for finding the starting point for each line (initialization) becomes too high. Therefore for the larger primes a different method has to be used. By this method it is comparably easy to compute a walk through all the points where the value $\log(p)$ has to be added. An efficient variant is described in [10]. After the sieving for (q, r) is finished, it is restarted with a different special q.

3.2.4 Generating the Matrix

This step is comparably fast and easy, even though for a 1024-bit number it might well take several hundred days. It is done with general-purpose computers, interfacing the special hardware or general-purpose computers used for the sieving and for the matrix step. It involves transferring, analysing and modifying a huge amount of data, but it does not need much computation.

Each row of the matrix M corresponds to a prime ideal. The upper rows correspond to prime ideals in the number field $\mathbb{Q}[x]/(f_0(x))$ (which usually is just $\mathbb{Q}$), the lower rows correspond to prime ideals in the number field $\mathbb{Q}[x]/(f_1(x))$. A column of M corresponds not only to a pair $(a, b) \in \mathbb{Z}^2$, but more precisely to a prime ideal decomposition of $F_0(a, b)$ in $\mathbb{Q}[x]/(f_0(x))$ (upper part of the column) and a prime ideal decomposition of $F_1(a, b)$ in $\mathbb{Q}[x]/(f_1(x))$ (lower part of the column).

Since the matrix M should not be too large, there should not be too many different prime ideals in the prime ideal decompositions, i.e., the $F_i(a, b)$ should only have small prime factors and therefore be smooth. This has been taken care of by the sieving.

Since the aim is to construct squares $\alpha^2 \equiv \beta^2 \pmod{N}$ from the $F_i(a,b)$, in the prime ideal decompositions only factors with odd exponents are relevant. In fact, just the parity of the exponents is needed. Therefore the matrix M contains only zeroes and ones. The matrix step consists in finding some solutions v of the equation $M \cdot v = 0$ in the vector space over $\mathbb{F}_2$.

First, depending on the sieving step, some columns might appear twice or more times. Duplicates have to be dropped such that every column appears just once. The matrix can be simplified before starting the matrix step in order to get a small matrix. Lines with all zeroes are dropped. If there is a line with exactly one "1", the line and the column of the "1" are dropped. This is called "pruning".

Filtering corresponds to some very careful Gaussian elimination. If a line contains exactly two entries "1", one column can be replaced by the sum of the two columns and then pruning can be done. For more entries "1" in a line one has to be careful to not increase the total number of ones in the matrix too much. Some sophisticated methods have been used for the present factoring records.

In practice, in the beginning the number of columns in the matrix (number of sieving reports) can be a bit smaller than the number of rows, as long as it is larger after pruning and filtering. If the resulting matrix is too large for the matrix step, it helps to collect more sieving reports, since pruning and filtering might then result in a smaller matrix.

3.2.5 Matrix Step

For large N this step should probably be done with special hardware. The present factoring records used general-purpose computers for this step. It took roughly 20% of the computing time for the whole factoring process.

The matrix step consists in finding enough solutions v of the equation $M \cdot v = 0$ in the vector space over $\mathbb{F}_2$. For a number N with 1024 bit the matrix might have a length of 10^{10} with perhaps $2 \cdot 10^{12}$ entries. These numbers are very uncertain and can only be guessed based on experiments with smaller numbers N. Since M does not have many non-zero entries, it is possible to store it in a compressed form.

There are several specialized algorithms for solving the equation. One can use ideas of the Block-Lanczos and the Look-ahead-Lanczos method or the Block-Wiedemann method (see [6], [25] and [36]). In these algorithms the many multiplications of M with different vectors are very expensive. Therefore, the part that has to be implemented in hardware is an efficient matrix by vector multiplication.

3.2.6 Square Root Step

This step is done with general-purpose computers within a few hours, there is no need for special hardware.

Each solution from the matrix step corresponds to a set $\{(a_i, b_i)\}$ of sieving reports such that the prime ideal decomposition in the number field $\mathbb{Q}[\mu_s]/f_s(\mu_s)$ of $q_s := \prod_i (a_i + b_i\mu_s)$, $s = 1, 2$, only contains even exponents. If the 2-rank of the ideal class group and the rank of the units of the number field are not both zero, q_s itself does not need to be a square. But using quadratic character checks it is possible to combine the solutions from the matrix step to give elements q_s which are squares in the number field.

The main computation in this step is calculating the square root r_s of q_s in the number field $\mathbb{Q}[\mu_s]/f_s(\mu_s)$. The preferred method has been developed by Montgomery (see [26] and [27]). Let $m \in \mathbb{Z}$ be the common zero modulo N of the two polynomials f_0 and f_1. Now α is obtained by replacing μ_0 with m in r_0 and β is obtained by replacing μ_1 with m in r_1. Then $\alpha^2 \equiv \beta^2 \pmod{N}$, i.e., N divides the product $(\alpha+\beta)(\alpha-\beta)$. With a heuristic probability of $\frac{1}{2}$, the number $\gcd(N, \alpha - \beta)$ is a non trivial factor of N. Checking many q_s eventually will lead to one factor (or all factors) of N.

3.3 Hardware supporting GNFS

There are a lot of choices for the implementation of GNFS in terms of algorithms, parameters, optimization tricks, and other details. The choice of parameters changes a lot when one switches between crackers for different bit lengths (in software as well as in hardware) and the whole strategy has to be adapted. Software experiments and simulations help a lot.

The field of integer factorization with the help of special-purpose hardware attracted more and more attention since 1999 (see e.g. [1], [29], [31], [14], [15], [24], [23]). This was reflected by a majority of contributions on that topic during the SHARCS Workshop (see [30], which gave a quite comprehensive summary of the state of the art of that specific but productive area. With sucessful integer factorization one can crack RSA moduli. Still RSA is used in many cryptographic protocols and in most crypto products. Therefore it is of permanent interest for the industry, government authorities and for users to get a good feeling about the actual security of the algorithms. Decisions about bit lengths are based on conclusions from cost estimates found in the cryptographic literature.

GNFS can also be used for solving the Discrete Logarithm problem in $\mathbb{F}_p$. In fact the sieving step for DL is basically identical to the sieving step for integer factorization, so in general the same hardware can be used. The matrix step then needs to solve a matrix modulo $p - 1$ instead of modulo 2. The other steps for solving the DL problem can be done with general-purpose computers.

3.4 Twinkle and Twirl

The TWINKLE sieving device was designed for numbers of the order of 512 or 768 bit and has been proposed in 1999 (see [29]). It consists of a wafer with a huge number of flashing LEDs. Each LED is in charge of a progression and emits light of intensity $\log(p)$ if p is in the progression. Each clock cycle corresponds to a sieving position. The global summation $\sum_i p_i$ is done by measuring the total amount of emitted light at each clock cycle. TWINKLE needs to be supported by auxiliary computation, e.g. a considerable amount of general-purpose

computers, which is the bottleneck for factoring larger numbers. It offers a relatively modest improvement over traditional sieving and heat dissipation is a major problem.

The TWIRL sieving device is a purely electronic device processing thousands of sieve locations at each clock cycle (proposed in 2003 by Shamir and Tromer, see [31]). Using the 130nm process technology, TWIRL consists of ASICs of the size of a full wafer (for the algebraic sieve) and the cost is 10 million US$ $\times$ year. Using the 90nm process technology, the cost of TWIRL is 1.1 million US$ $\times$ year. It is not clear how to create working ASICs of the desired size.

TWIRL implements line sieving by splitting the line in chunks of length 4096 for the rational sieve. The sum of the contributions $\log(p)$ is calculated by a pipeline of adders at all 4096 locations in parallel using a 4096×10-bit bus. The sieve locations flow down the bus in steps of 4096. The sieving contributions travel horizontally across the bus and reach an appropriate position at exactly the right time. There are stations that perform the scheduling and routing. The primes of the factor base cover a very large range of sizes and different sizes require different design tradeoffs. Therefore there are three different types of stations: for the small, the medium and for the large primes. The large primes are the most difficult to handle since they are numerous. Each large prime gives only few contributions, but every contribution is large and thus important.

The needed bandwidth both along the pipeline between the stations and across the pipeline is enormous. Thus TWIRL is inherently a one wafer design, since trying to divide the wafer in smaller ASICs would imply a considerably reduced performance due to the limited throughput of the connections between the ASICs. The parameters of TWIRL are not optimal, since the ASIC for the algebraic sieve has to fit on a wafer. Therefore when the technology will be advanced enough to build TWIRL, further progress would allow for better parameters, and finally TWIRL would get cheaper at a faster rate than what is suggested by Moore's law.

3.5 SHARK

Proposed in 2005 (see [12]), the design of the SHARK sieving device focuses on realizability with today's technology. It has a modular architecture of small ASICs connected mainly via standard buses. It consists of 2300 identical isolated machines sieving in parallel. Each machine costs 70 000 US$. SHARK performs the sieving step of GNFS for a 1024-bit number within a year and costs around 200 million US$ (this includes 40 million US$ for the power consumption for one year).

In contrast to other machines, SHARK uses lattice sieving which, for large numbers N, is more efficient than line sieving. The actual sieving is done in very fast accessible memory ("cache"), 32 MB per machine. The total sieving area is split into parts of this size. Now every sieving contribution of each prime of the two factor bases has to be transported to the sieving memory at the correct time. This is done in three different ways, depending on the size of the prime.

The sieving contributions of the large primes are generated in 1024 units and then dis-

tributed globally to one of 1024 positions through a butterfly transport system. At each of these 1024 end points these data are going to be processed and output locally. Therefore the machine splits into 1024 identical branches without interconnection, so that the machine has 1024 outputs. The following describes one of these branches. The sieving contributions of the large primes are combined with the sieving contributions of the medium primes, which are generated locally in this branch. The evaluator is the largest ASIC (maybe the size of a Pentium processor) which collects and evaluates the result from the sieving cache. It combines this result with the sieving contributions of the small primes, generated on the fly in a small unit. The potential sieving reports still have to be checked for smoothness by an ECM unit (see Section 3.6).

The potential sieving reports that pass the ECM smoothness test are collected by a general-purpose computer from each of the 1024 outputs. There is still need for some general-purpose computer for controlling the sieving process, i.e. feeding the machine with data. The most expensive part of the machine is the 136 GB DRAM which is needed in the different parts of the machine in quantities between 8 MB and 64 MB. At a clock frequency of 1 GHz one machine takes around 20 seconds per special q, such that 2300 machines can compute $3.7 \cdot 10^9$ special q and complete the sieving step within a year. The power consumption of one machine is estimated (considering the ASIC area) to 30 kW.

In order to justify the realizability with today's technology, the most critical part seems to be the butterfly transport system, which performs the task of a switch with 1024 inputs and 1024 outputs. More precisely, its purpose is to transport data chunks of 80 bits from one input to an output determined by 10 of these 80 bits. These chunks may arrive simultaneously at different input channels. To avoid collisions, there should be some buffers, but a small loss of data is tolerable. The latency of the transport system is not important, but the throughput has to be at peak times at least 1 GB per second per input channel, $\frac{1}{5}$ of this on average.

The SHARK design is not optimized for efficiency, since this might make the claim about the realizability with today's technology more questionable. Using more ECM, changing several parameters and enlarging the butterfly transport system might reduce the overall cost considerably.

3.6 ECM in Hardware

As discussed in the beginning of Section 3.1, the General Number Field Sieve is the best known algorithm for factoring numbers with large factors (hundreds of bits) and, hence, can be used for attacking the RSA cryptosystem. As an intermediate step, GNFS requires a method to efficiently factor lots of smaller numbers (factorization of cofactors). An appropriate choice for this task is the Multiple Polynomial Quadratic Sieve (MPQS) or the Elliptic Curve Method (ECM, see [21]).

It appears that the use of ECM rather than MQPS is the better choice for a hardware realization, since MQPS requires a larger silicon area and irregular operations. On the other hand, ECM is an almost ideal algorithm for dramatically improving the time-cost product

through special-purpose hardware. First, it performs a very high number of operations on a very small set of input data, and is, thus, not very I/O intensive. Second, it requires relatively little memory. Third, the operands needed for supporting GNFS are well beyond the width of current computer buses, arithmetic units, and registers, so that special-purpose hardware can provide a much better fit. Lastly, it should be noted that the nature of cofactorization in GNFS allows for a very high degree of parallelization. Hence, the key for efficient ECM hardware lies in fast arithmetic units which have been studied thoroughly in the last few years, e.g., for the use in cryptographic devices using Elliptic Curve Cryptography (ECC).

[13] presents an efficient hardware implementation of ECM to factor numbers up to 200 bits, which is also scalable to other bit lengths. For proof-of-concept purposes, ECM has been realized as a software-hardware co-design on an FPGA and an embedded microcontroller. The design has a good scalability to larger and smaller bit lengths. In some range, both the time and the silicon area depend linearly on the bit length. One should notice that while direct application of ECM to factor numbers with length of several hundred bits is much less efficient than GNFS and the method is asymptotically slow, ECM is highly suitable for effective processing of the smoothness testing step within GNFS that requires factorization up to 200-bit operands.

3.6.1 The Algorithm

The principles of ECM are based on Pollard's $(p-1)$-method [28]. Now, we give a brief description of the Elliptic Curve Method (ECM). For more details on ECM see [21].

Let N be an integer without small prime factors which is divisible by at least two different primes, one of them q. Such numbers appear after trial division and a quick prime power test. Let $E/\mathbb{Q}$ be an elliptic curve with good reduction at all prime divisors of N (this can be checked by calculating the gcd of N and the discriminant of E, which very rarely yields a prime factor of N) and a point $P \in E(\mathbb{Q})$. Let $E(\mathbb{Q}) \to E(\mathbb{F}_q), Q \mapsto \overline{Q}$ be the reduction modulo q. Let B_1 and B_2 two bounds. If the order o of $\overline{P} \in E(\mathbb{F}_q)$ satisfies certain smoothness conditions described below, we can discover the factor q of N as follows:

- *Phase 1*: Calculate $Q = kP$, where $k = \prod_{p \leq B_1 \text{ prime}} p^{e_p}$ and $e_p = \left\lfloor \frac{\log B_1}{\log p} \right\rfloor$.
- *Phase 2*: Check for each prime $B_1 < p \leq B_2$ whether pQ reduces to the neutral element in $E(\mathbb{F}_q)$. This can be done, e.g. using the Weierstraß form and projective coordinates $pQ = (x_{pQ} : y_{pQ} : z_{pQ})$, by testing whether $\gcd(z_{pQ}, N)$ is bigger than 1.

All calculations are done modulo N. If an inversion is not possible (e.g., using affine coordinates in the Weierstraß form) a divisor is found. More precisely, if o factors into a product of coprime prime powers (each $\leq B_1$) and at most one additional prime less than another bound B_2 the prime factor q is discovered. Otherwise the procedure will be repeated for other elliptic curves. To generate them one commences with the starting point P and constructs an elliptic curve such that P lies on it. Note that we can avoid all gcd computations but one at the expense of one modular multiplication per gcd by accumulating the numbers to be checked in a product modulo N and doing one final gcd.

Apart from the Weierstraß form there are various other forms for the elliptic curves. We use Montgomery's form ($By^2z = x^3+Ax^2z+xz^2$) and compute in the set $S = E(\mathbb{Z}/N\mathbb{Z})/\{\pm 1\}$ only using the x- and z-coordinates.

For the computation of both phases of ECM, sophisticated algorithms which are especially well suited for a hardware implementation are used. In phase 1, an efficient point multiplication is achieved by a step-wise point multiplication with adjacent primes. Phase 2 continues phase 1 by efficiently using a small table of precomputed point multiples. This variant of the improved standard continuation reduces the memory requirements and, thus, can be implemented with low area consumption.

The actual parameterization of the ECM unit was deduced from software experiments. Let $B_1 = 960$ and $B_2 = 57000$. For the chosen parameters, the computational complexity of phase 1 is 13740 modular multiplications and squarings. A total of 11 registers for storing intermediate values is required for phase 1. For phase 2, the parameters are chosen to keep the size of the table low at the cost of an increased running time. Only 8 additional registers are required to store all coordinates in the table. The total computational complexity of phase 2 is 24926 modular multiplications and squarings. For finding a factor with at most 42 bits with a probability of $> 80\%$, software experiments showed for the parameters given that we need to run ECM on approximately 20 different curves.

For a probability of finding a factor of $> 80\%$, software experiments showed for the parameters given that we need to run ECM on approximately 20 different curves.

3.6.2 Hardware Architecture

The ECM unit mainly consists of three parts: the ALU (arithmetic logic unit), the memory part (registers) and an internal control logic. Each unit has a very low communication overhead with a central control logic since all input operands and results are stored in the unit during computation. A central control logic is connected to each ECM unit and coordinates the data exchange with the unit before and after computation and starts each computation in the unit by a special set of commands stored in a control register. If several ECM units work in parallel, only one central control register is needed. All commands are sent in parallel to all units and computations start at the same time. Only in the beginning and in the end, the unit's memory cells have to be written and read out separately.

Each unit possesses an internal control logic in order to coordinate the data in- and output from and to the registers, respectively, and to control the computation process of the ALU. The arithmetic is coordinated inside the ALU and communicated with the internal control logic.

The ALU performs the basic modular arithmetic operations. The core operations, namely modular multiplication and modular squaring, have been realized with a highly efficient variant of a pipelined systolic Montgomery multiplier. The pipelining depth and the number of processing elements can be modified and allow for shifting the area and time complexity of the design. Details on the choice of the implemented algorithms can be found in [13].

3.6.3 Parallelization

ECM can be perfectly parallelized by using different curves in parallel since the computations of each unit are completely independent. For the control of more than one ECM unit, it is essential to know that both phases, phase 1 and phase 2, are controlled completely identically, independent of the composite to be factored. Solely the curve parameter and possibly the modulus of the units and, hence, the coordinates of the initial point differ. Thus, all units have to be initialized differently which is done by simply writing the values into the corresponding memory locations sequentially. The required time for this data IO is negligible for one ECM unit since the computation time of both phases dominates. For several units in parallel, the computation time does not change, but the time for data IO scales linearly with the number of units. Hence, not too many units should be controlled by one single logic. For massively parallel ECM in hardware, the ECM units can be segmented into clusters, each with its own control unit.

3.6.4 Implementation

The ECM implementation in [13] is a hybrid design. It consists of an ECM unit implemented on an FPGA (Xilinx Virtex2000E-6) and a control logic implemented in software on an embedded microcontroller (ARM7TDMI, 25MHz). The ECM unit is coded in VHDL and was synthesized for a Xilinx FPGA (Virtex2000E-6). The design was done for $n = 198$ bit composites. The whole design is fully scalable and bit lengths from 100 to 300 bits can be easily accomplished. In this case, the AT product will de-/ increase according to the size of n^2.

Phase 1 was accomplished in 912 ms, phase 2 in 1879 ms. The ECM unit including the full support for the phase 1 and 2 of ECM with a word width of 32 bits and 2 pipeline stages has the following area requirements: 1754 lookup-tables (LUT), 506 flip-flops and 44 Blocks RAM. Minimum clock period is 26ns. Since the design was done for proof-of-concept purposes, further improvements in data configuration and better management of input and output operands for the ALU inside the ECM unit will yield higher performance.

As demonstrated, ECM can be perfectly parallelized and, thus, an implementation at a larger scale can be used to assist the GNFS factoring algorithm by carrying out all required smoothness tests. The concept of the GNFS architecture SHARK counts with around 3.7 millions of the ECM units. Using many more ECM units, a low cost ASIC implementation of ECM can decrease the overall costs of SHARK, as shown in [12].

3.7 Matrix Step in Hardware

The first proposals for solving the matrix of GNFS in special hardware can be found in [1] and [24]. Considerably cheaper solutions are given in [16] and [17], but the technological hurdles are comparable to those for TWIRL: extremely large chips, many high-bandwidth chip interconnections.

The design of [18] is more efficient and more realistic. It consists of a one or two dimensional chain of identical chips with standard chip sizes and only local interconnections. The cost for solving the matrix is 400 000 US$×year. The design can also be used for matrices

over $\mathbb{F}_p$.

In contrary to the sieving step, the matrix step does not tolerate computational errors at all. Because of the huge amount of data and the long computation time, errors induced by hardware glitches or radiation are unavoidable. Therefore an error correction mechanism is indispensable. In the full version of [18] this problem is solved.

Bibliography

[1] D. J. Bernstein, *Circuits for Integer Factorization: A Proposal*, Manuscript, November 2001.
http://cr.yp.to/papers.html#nfscircuit

[2] D. J. Bernstein, *Integer factorization: a progress report*, Talk at FoCM 2005.
http://cr.yp.to/talks.html

[3] R. P. Brent, *Recent Progress and Prospects for Integer Factorisation Algorithms*, COCOON 2000, LNCS **1858**, Springer, 2000, 3-22.

[4] H. Cohen, *A Course in Computational Algebraic Number Field Theory*, Graduate Texts in Math. **138**, Springer, 1993.

[5] S. Bajracharya, D. Misra, K. Gaj, T. El-Ghazawi, *Reconfigurable Hardware Implementation of Mesh Routing in the Number Field Sieve Factorization*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[6] D. Coppersmith, *Solving homogeneous linear equations over $GF(2)$ via block Wiedemann algorithm*, Math. Comp. **62** (1994), 333-350.

[7] P. Zimmermann, ECMNET, Website, 2005.
http://loria.fr/~zimmerma/records/ecmnet.html

[8] ECRYPT NoE, *ECYRPT Report on Hardness of the Main Computational Problems used in Cryptography*, Document D.AZTEC.4, 2005.

[9] J. Franke, T. Kleinjung et al., *RSA*-576, E-mail announcement, 2003.
http://www.crypto-world.com/announcements/rsa576.txt

[10] J. Franke, T. Kleinjung, *Continued Fractions and Lattice Sieving*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[11] J. Franke, T. Kleinjung et al., *RSA*-200, E-mail announcement, 2005.
http://www.crypto-world.com/announcements/rsa200.txt

[12] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, C. Stahlke, *SHARK — A Realizable Special Hardware Sieving Device for Factoring* 1024*-bit Integers*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005, also to appear in Proc. CHES 2005.

[13] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, M. Šimka, C. Stahlke, *An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[14] W. Geiselmann and R. Steinwandt, *A dedicated sieving hardware*, PKC 2003, LNCS **2567**, Springer, 2003, 254-266.

[15] W. Geiselmann and R. Steinwandt, *Yet another sieving device*, CT-RSA 2004, LNCS **2964**, Springer, 2004, 278-291.

[16] W. Geiselmann and R. Steinwandt, *Hardware for Solving Sparse Systems of Linear Equations over $GF(2)$*, in: Proc. CHES 2003, LNCS **2779**, Springer, 2003, 51-61.

[17] W. Geiselmann, H. Köpfer, R. Steinwandt and E. Tromer, *Improved Routing-Based Linear Algebra for the Number Field Sieve*, in: Proc. ITCC 2005 - Track on Embedded Cryptographic Systems, IEEE Computer Society, 2005.

[18] W. Geiselmann, A. Shamir, R. Steinwandt, E. Tromer, *A systolic design for supporting Wiedemann's algorithm*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[19] R. A. Golliver, A. K. Lenstra and K. S. McCurley, *Lattice sieving and trial division*, in: Algorithmic Number Theory (ed. by L. M. Adleman, M.-D. Huang), LNCS **877**, Springer, 1994, 18–27.

[20] T. Kleinjung, *On Polynomial Selection for the General Number Field Sieve*, 2004, to appear in Mathematics of Computation.

[21] H. W. Lenstra, *Factoring Integers with Elliptic Curves*, Annals of Mathematics **126 no. 2**, 1987, 649-673.

[22] A.K. Lenstra and H.W. Lenstra, Jr. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. **1554**, Springer, 1993.

[23] A. K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes and P. Leyland, *Factoring Estimates for a* 1024*-bit RSA Modulus*, in: Proc. ASIACRYPT 2003, LNCS **2894**, Springer, 2003, 55-74.

[24] A. K. Lenstra, A. Shamir, J. Tomlinson und E. Tromer, *Analysis of Bernstein's Factorization Circuit*, Preprint, 2002.
`http://www.wisdom.weizmann.ac.il/~tromer/papers/meshc.ps.gz`

[25] P. L. Montgomery, *A Block Lanczos Algorithm for Finding Dependencies over $GF(2)$*, 106-120, in: Advances in Cryptology: Eurocrypt '95, LNCS **921**, Springer, 1995.

[26] P. L. Montgomery, *Square roots of products of algebraic numbers*, 1995.
`ftp://ftp.cwi.nl/pub/pmontgom/sqrt.ps.gz`

[27] P. Nguyen, *A Montgomery-Like Square Root for the Number Field Sieve*, Alg. Number Theory – ANTS III, LNCS **1443**, 1998, 151-168.

[28] J. M. Pollard, *A Monte Carlo Method for Factorization*, Nordisk Tidskrift for Informationsbehandlung (BIT) **15**, 1975, 331-334.

[29] A. Shamir, *Factoring large numbers with the TWINKLE device (extended abstract)*, in: Proc. CHES 1999, LNCS **1717**, Springer, 1999, 2-12.

[30] *SHARCS 2005 – Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems*, ENSTA Paris, Booklet, Februar 2005.
`http://www.sharcs.org`

[31] A. Shamir and E. Tromer, *Factoring Large Numbers with the TWIRL Device*, in: Proc. Crypto 2003, LNCS **2729**, Springer, 2003, 1–26.
`http://www.wisdom.weizmann.ac.il/~tromer/papers/twirl.ps.gz`

[32] A. Shamir and E. Tromer, *Special-purpose Hardware for Factoring: the NFS Sieving Step*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[33] T. Shimoyama, T. Izu, J. Kogure, *Implementing a Sieving Algorithm on a Dynamic Reconfigurable Processor*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005.

[34] E. Tromer, *Special-Purpose Cryptanalytic Devices*, Website on Special-Purpose Hardware for Cryptanalysis, 2005.
`http://www.wisdom.weizmann.ac.il/~tromer/cryptodev/`

[35] M. J. Wiener, *The Full Cost of Cryptanalytic Attacks*, Journal of Cryptology **17**, 2004, 105-124.

[36] D. H. Wiedemann, *Solving Sparse Linear Equations over Finite Fields*, IEEE Transactions on Information Theory **32(1)**, 1986, 54-62.

Chapter 4

DES Crackers

4.1 History

People have questioned the security of the DES [10] for a long time and there has been much speculation on its design principles, *e.g.* for the cryptographic significance of the S-boxes. However, in practice, no other attack has been as intensively discussed (and demonstrated) as exhaustive key search. Even nowadays, the major concern about the security of the DES is its too short key length. In this section, we briefly review certain historical results of exhaustive key search machines.

The first exhaustive DES key search machine estimation was proposed by Diffie and Hellman in 1977 [6] and contained 10^6 DES chips, with an estimated cost of US$ 20M (Million) and a 12-hour expected search time. They argued that this was out of reach for almost everybody, excepted organizations like the National Security Agency (NSA), but that by the 1990s, the DES would be totally insecure. Meanwhile, hardware implementations of the DES slowly approached the million-encryption-per-second requirement of Diffie and Hellman's special-purpose machine. By 1987, chips performing 512 000 encryptions per second were being developed.

Subsequently, Quisquater and Delescaille studied the related question of collision search in the DES in [11], while Quisquater and Desmedt investigated the cost of a random key search machine compared to a systematic search in a table in [12]. They also suggested that distributed computing could be a solution for such computationally intensive problems.

In 1993, Wiener [19] provided a gate-level design for a US$ 1M machine using 57 600 DES chips with an expected success in 3.5 hours. Every chip contained 16 pipeline stages, running at a clock frequency of 50 Mhz, going on with the popular story of "DES crackers".

At the 1997 annual RSA Cryptographic Trade Show in San Francisco, a prize was announced for cracking a DES cryptogram. The prize was claimed in five months, by a loose consortium using computers scattered around the Internet. It was the most dramatic success so far for an approach earlier applied to factoring and to breaking cryptograms in systems with 40-bit keys. At the 1998 RSA show, the prize was offered again. This time, it was claimed in 39 days.

Finally, to prove the insecurity of the DES, the Electronic Frontier Foundation (EFF) built the first unclassified hardware for cracking DES. On Wednesday, July 17, 1998, the EFF DES Cracker [5], which was built for less than US$ 200 000, easily won the RSA Laboratories's "DES Challenge II " contest. It took the machine less than 3 days to complete the challenge, shattering the previous record of 39 days set by a massive network of ten thousand computers.

However, this fulfilled challenge did not stop the investigations about exhaustive key search machines. In the following, we will describe more recent attempts to defeat the DES by brute force techniques. First, from a technological point of view, Section 4.2 discusses FPGA (Field Programmable Gate Array) machines able to recover a 56-bit DES key and evaluates their possible cost. Second, Section 4.3 surveys the possible time-memory tradeoff attacks against block ciphers and their application to DES.

4.2 Exhaustive key search using FPGAs

Due to its potential to greatly accelerate a wide variety of applications while maintaining good flexibility, reconfigurable computing has gained importance in the industrial development of digital signal processing applications. FPGAs notably allowed to develop extremely fast and compact encryption architectures of various block ciphers including the DES, *e.g.* in [14]. These designs have been used to perform the fastest-known experimental linear cryptanalysis of the DES in [15].

It is usually considered that the major problem of current reconfigurable devices is their high individual cost. Latest Xilinx Virtex® devices can presently cost up to US$ 1 000. However, regarding cheaper technologies, Xilinx® recently announced (October 6, 2003) breakthrough price points for its low cost Spartan-3® family. This low cost, in addition to the usual advantages of FPGAs compared to ASICs (Application Specific Integrated Circuits), namely (1) less engineering expertise required and (2) lower initial development costs, make these reconfigurable devices an interesting alternative for exhaustive key search machines. As an illustration, table 4.1 compares some implementation results of a 37 pipeline stages unrolled DES for these two technologies.

Device	**Virtex-II®**	**Spartan-3®**
Nbr of LUTs	3775	3780
Nbr of registers	4387	4408
Nbr of slices	2965	2958
Clock frequency (Mhz)	333	180
Encryption rate (Gbits/sec)	21.3	11.5

Table 4.1: DES implementation results.

Regarding price constraints, the 3S1000 Spartan-3® device with 1 million system gates is available for under US$ 12. It contains 7 680 slices and can consequently embed 2 DES designs. Considering a realistic clock frequency of 134 Mhz, we may therefore assume an encryption rate of $2 \times 134.10^6 \simeq 2^{28}$ encryptions per second for US$ 12.

These predictions already suggest that the use of recent (low cost) reconfigurable devices can be relevant for exhaustive key search attacks against block ciphers. As a practical estimation, a US$ 12 000 machine could break DES in about 3 days, presently or in the near future. Although these estimations do not take the development costs into account and must therefore be relativized, they clearly underline that the cost of a DES cracker is presently reachable for most organizations, including universities.

The design of an exhaustive key search machine from any efficient block cipher implementation is straightforward. It is only required to have the opportunity to change the key easily, which is usually the case for most of the DES implementations. As a consequence, we do not give more details about such designs and note that the construction of an FPGA-based DES cracker is planned within ECRYPT (at the University of Bochum).

4.3 Time-memory tradeoffs

Many searching problems allow time-memory tradeoffs. That is, if there are K possible solutions to search over, the time-memory tradeoff allows the solution to be found in T operations (time) with M words of memory, provided the time-memory product $T \times M$ equals K. Cryptanalytic attacks based on exhaustive key search are the typical context where time-memory tradeoffs are applicable.

Due to large key sizes, exhaustive key search usually needs unrealistic computing powers and corresponds to a situation where $T = K$ and $M = 1$. However, if the same attack has to be carried out numerous times, it may be possible to execute the exhaustive search in advance and store all the results in a memory. Once this precomputation is done, the attack could be performed almost instantaneously, although in practice, the method is not realistic because of the huge amount of memory needed: $T = 1$, $M = K$. The aim of a time-memory tradeoff is to mount an attack that has a lower online processing complexity than exhaustive key search and lower memory complexity than a table lookup. The method can be used to invert any one-way function and was originally presented by Hellman in [6].

4.3.1 The original method

Let $E_K(\mathrm{X}) : 2^n \times 2^k \rightarrow 2^n$ denote an encryption function of a n-bit plaintext X under a k-bit secret key K. The time-memory tradeoff method needs to define a function g that maps ciphertexts to keys: $g : 2^n \rightarrow 2^k$. If $n > k$, g is a simple reduction function that drops some bits from the ciphertexts (e.g. in the DES, $n = 64$, $k = 56$). If $n < k$, g adds some constant bits. Then we define

$$f(K) = g(E_K(P)), \tag{4.1}$$

where P is a fixed chosen plaintext. Computing $f(K)$ is almost as simple as enciphering, but computing K from $f(K)$ is equivalent to cryptanalysis. The time-memory tradeoff method is composed of a precomputation task and an online attack that we describe as follows.

Precomputation task: The cryptanalyst first chooses m different start points: SP_1,

SP_2,..., SP_m from the key space. Then he computes encryption chains where $X_{i,0} = SP_i$ and $X_{i,j+1} = f(X_{i,j})$, for $1 \leq j \leq t$:

$$\begin{array}{l} X_{0,0} \xrightarrow{f} X_{0,1} \xrightarrow{f} X_{0,2} \xrightarrow{f} \ldots\ldots\ldots \xrightarrow{f} X_{0,t} \\ X_{1,0} \xrightarrow{f} X_{1,1} \xrightarrow{f} X_{1,2} \xrightarrow{f} \ldots\ldots\ldots \xrightarrow{f} X_{1,t} \\ X_{2,0} \xrightarrow{f} X_{2,1} \xrightarrow{f} X_{2,2} \xrightarrow{f} \ldots\ldots\ldots \xrightarrow{f} X_{2,t} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots\ldots\ldots \\ X_{m,0} \xrightarrow{f} X_{m,1} \xrightarrow{f} X_{m,2} \xrightarrow{f} \ldots\ldots\ldots \xrightarrow{f} X_{m,t} \end{array} \tag{4.2}$$

To reduce the memory requirements, the cryptanalyst only stores start and end points ($SP_i = X_{i,0}$, $EP_i = X_{i,t}$) and sorts the $\{SP_i, EP_i\}_{i=1}^{m}$ on the end points. The sorted table is stored as the result of this precomputation.

Online attack: Now we assume that someone has chosen a key K and the cryptanalyst intercepts or is provided with $C = E_K(P)$. Then he can apply the function g to obtain $Y = g(C) = f(K)$ and follows algorithm 1.

Algorithm 1 Online attack

1. If $Y = EP_i$, then either $K = X_{i,t-1}$ or EP_i has more than one inverse image. We refer to this latter event as a false alarm. If $Y = EP_i$, the cryptanalyst therefore computes $X_{i,t-1}$ and checks if it is the key, for example by seeing if it deciphers C into P
2. If Y is not an end point or a false alarm occurred, the cryptanalyst computes $Y = f(Y)$ and restarts step 1.

Remark that the cryptanalyst needs to access the table lookup every time a new Y is computed. If all $m \times t$ elements are different, the probability of success PS would be $\frac{m \times t}{2^k}$. The actual probability of success depends on how the precomputed chains cover the key space. Unfortunately, there is a chance that chains starting at different keys collide and merge. The larger is a table, the higher is the probability that a new chain merges with a previous one. Each merge reduces the number of distinct keys that are actually covered by the table. If f is a random function, then the probability of success is bounded by:

$$PS_{table} \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} (1 - \frac{it}{N})^{j+1} \tag{4.3}$$

Equation 4.3 indicates that, for a fixed value of N, there is not much to be gained by increasing m or t beyond the point at which $mt^2 = N$. To obtain a high probability of success, a more efficient method consists in generating multiple tables using a different function g for each table. The probability of success with r tables is:

$$PS_{tot} \geq 1 - (1 - PS_{table})^r \tag{4.4}$$

Chains of different tables can collide, but not merge since the function g is different for every table.

4.3.2 Distinguished points and rainbow tables

The idea of using distinguished points (DPs) in time-memory tradeoffs refers to Rivest in [4]. If $\{0,1\}^k$ is the key space, a DP property of order d is usually defined as an easily checked property that holds for 2^{k-d} different elements of $\{0,1\}^k$, e.g. having d bits of the key locked to zero. In a time-memory tradeoff using DPs, the start and end points of the precomputed chains fulfill a DP property. As a consequence, the chains have variable length but detectable extreme points. The resulting tradeoff using distinguished point is similarly composed of a precomputation task and online attack which are intuitively represented in Figures 4.1 and 4.2.

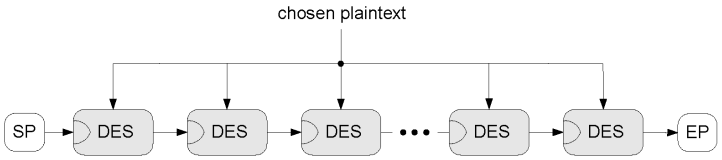

Figure 4.1: Precomputation task.

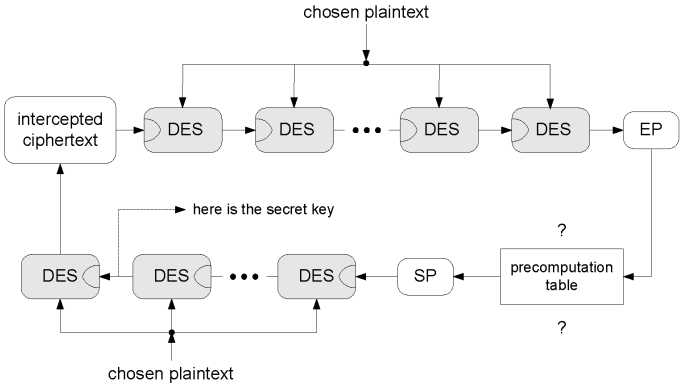

Figure 4.2: Online attack.

Compared to the original tradeoff, this solution greatly reduces the number of table lookups during the online attack from t to 1. Another remarkable property of the DP method is that mergers can be easily detected and therefore, can possibly be rejected during the precomputation in order to build perfect tables [2]. The major drawback of DPs is that they introduce variable chain lengths and are more difficult to analyze [17].

An alternative solution to reduce the number of table lookups is to use the rainbow tables

presented in [9]. That is to use a different function g for each point in a chain:

$$\begin{array}{l} X_{0,0} \stackrel{f_1}{\rightarrow} X_{0,1} \stackrel{f_2}{\rightarrow} X_{0,2} \stackrel{f_3}{\rightarrow} ..... \stackrel{f_t}{\rightarrow} X_{0,t} \\ X_{1,0} \stackrel{f_1}{\rightarrow} X_{1,1} \stackrel{f_2}{\rightarrow} X_{1,2} \stackrel{f_3}{\rightarrow} ..... \stackrel{f_t}{\rightarrow} X_{1,t} \\ X_{2,0} \stackrel{f_1}{\rightarrow} X_{2,1} \stackrel{f_2}{\rightarrow} X_{2,2} \stackrel{f_3}{\rightarrow} ..... \stackrel{f_t}{\rightarrow} X_{2,t} \\ \qquad\qquad\qquad\qquad\qquad\qquad ..... \\ X_{m,0} \stackrel{f_1}{\rightarrow} X_{m,1} \stackrel{f_2}{\rightarrow} X_{m,2} \stackrel{f_3}{\rightarrow} ..... \stackrel{f_t}{\rightarrow} X_{m,t} \end{array} \tag{4.5}$$

Two rainbow chains can only merge if they collide at the same position. Other collisions do not provoke a merge. The method is extremely easy to analyze and one rainbow table may contain t times more chains than an original table.

As a consequence, rainbow tables are presently considered as the easiest and most efficient way to perform a time-memory tradeoff. DP methods have a more theoretical interest but may also be used to detect collisions (e.g. of hash function) as suggested in [12, 18].

4.3.3 Implementation

As it is possible to use efficient FPGA implementations of block ciphers to perform exhaustive key search, we can adapt a pipeline design in order to carry out the precomputation part of the time-memory tradeoff attack. In this section, we present a generic design for precomputing encryption chains.

In general, if the targeted block cipher is implemented as a p-stage pipeline design, we will deal with p different chains in parallel [16]. A simple solution to manage this task consists in storing the different start points (denoted as $K_1, K_2, \ldots, K_p$) in a p-stage shift register as represented in Figure 4.3. In this architecture, the block cipher output always corresponds to an intermediate point in the chain for which the start point is available at the shift register output. Every time a distinguished point is detected, we output the resulting end point and increment the corresponding start point in order to initialize a new chain. In addition, a different mask function (represented as a bitwise $\oplus$ with a constant Boolean vector in the figure) can be applied to every computed chain. Depending on the type of tradeoff implemented (original one, using DPs or rainbow tables), the generation of the masks has to be adapted. This design can obviously be parallelized for different mask functions. Finally, p additional counters are required in order to compute the different chain lengths, although it is not represented in Figure 4.3.

The major advantage of this scheme is that it only requires an efficient block cipher implementation and a few additional resources (*e.g.* counters). Moreover, due to the "shift register-pipeline" structure, it does not imply any complex interface to output the results and does not reduce the block cipher clock frequency. As a consequence, it allows the precomputation tables for time-memory tradeoffs to be computed at roughly the same cost and throughput as an exhaustive key search.

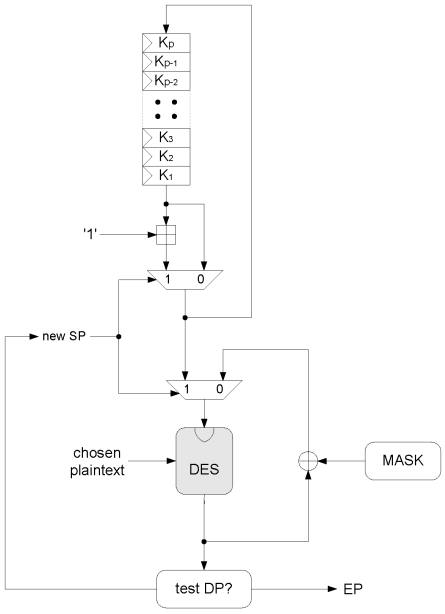

Figure 4.3: FPGA precomputation design.

4.3.4 Practical attacks

From a practical point of view, time-memory tradeoff attacks have recently been applied in different contexts. First, in [13], a time-memory tradeoff using distinguished points has been implemented using FPGAs against a 40-bit DES. The resulting online attack is reported to be feasible on any PC within about 10 seconds, with a success rate of 72%. A similar implementation of a tradeoff using rainbow tables has been investigated in the context of an attempt to break DES-based Unix passwords [8]. However, due to the complexity of the precomputation task, the passwords were only threatened with a weak probability. This attack has been improved very recently by Biryukov et al. [1]. In 2002, Clayton and Bond [3] described experiments to attack the IBM 4758 CCA® device, used in retail banking to protect the ATM infrastructure. This practical scheme collected the necessary data in a single 10-minute session. Finally, in 2003, a Swiss team from the EPFL[1] used a time-memory tradeoff device to implement an attack on MS-Windows® password hashes. Using 1.4 GB of data, they were able to crack 99.9% of all the alphanumerical passwords hashes (from a set of 2^{37} items) in 13.6 seconds.

[1]EPFL: Ecole Polytechnique Federale de Lausanne.

Bibliography

[1] Alex Biryukov, Sourav Mukhopadhyay and Palash Sarkar. Improved Time-Memory Trade-offs with Multiple Data In *Proceedings of SAC 2005*, to appear

[2] Johan Borst, Bart Preneel, and Joos Vandewalle. On the time-memory tradeoff between exhaustive key search and table precomputation. In *Proceedings of the 19th Symposium in Information Theory in the Benelux*, pages 111–118, 1998.

[3] Richard Clayton and Mike Bond. Experience using a low-cost fpga design to crack des keys. In Jr. et al. [7], pages 579–592.

[4] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[5] Electronic Frontier Foundation. *Cracking DES*. O'Reilly & Associates, 1998.

[6] Martin Hellman. A cryptanalytic time-memory tradeoff. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[7] Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2003.

[8] Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Cracking unix passwords using fpga platforms. In *Proceedings of SHARCS - Special-purpose Hardware for Attacking Cryptographic Systems*, 2005.

[9] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[10] U.S. Dept. of Commerce. Fips pub 46, the data encryption standard, federal information processing standard, 1977.

[11] Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search. new results and applications to des. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 408–413. Springer, 1989.

[12] Jean-Jacques Quisquater and Yvo Desmedt. Chinese lotto as an exhaustive code-breaking machine. *IEEE Computer*, 24(11):14–22, 1991.

[13] Jean-Jacques Quisquater, François-Xavier Standaert, Gaël Rouvroy, Jean-Pierre David, and Jean-Didier Legat. A cryptanalytic time-memory tradeoff: First fpga implementation. In Manfred Glesner, Peter Zipf, and Michel Renovell, editors, *FPL*, volume 2438 of *Lecture Notes in Computer Science*, pages 780–789. Springer, 2002.

[14] Gaël Rouvroy, François-Xavier Standaert, Jean-Jacques Quisquater, and Jean-Didier Legat. Design strategies and modified descriptions to optimize cipher fpga implementations: Fast and compact results for des and triple-des. In Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, editors, *FPL*, volume 2778 of *Lecture Notes in Computer Science*, pages 181–193. Springer, 2003.

[15] Gaël Rouvroy, François-Xavier Standaert, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient uses of fpgas for implementations of des and its experimental linear cryptanalysis. *IEEE Trans. Computers*, 52(4):473–482, 2003.

[16] François-Xavier Standaert. *Secure and Efficient Uses of Reconfigurable Hardware Devices in Symmetric Cryptography.* Université Catholique de Louvain, Phd thesis, 2004.

[17] François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. A time-memory tradeoff using distinguished points: New analysis & fpga results. In Jr. et al. [7], pages 593–609.

[18] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[19] Michael Wiener. Efficient des key search, technical report tr-244, school of computer science, carleton university, ottawa, 1994.

[20] J.-J. Quisquater, F.-X. Standaert, *Exhaustive Key Search of the DES: Updates and Refinements*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005 , Paris, 2005.

Chapter 5

Conclusions

In this report, we have surveyed different state-of-the-art techniques of building hardware crackers to attack the RSA and the DES algorithm.

The reason for investigating hardware crackers is the chance to build more efficient (cheaper and faster) crackers than would be possible in software and therefore to obtain a completely different quality of cracking power. The public interest in designing hardware crackers is on the one hand to get aware of the dangers of leaking information and on the other hand to get convincing estimates about the security of widely used algorithms and key lengths. Given the increasing number of applications depending on cryptographic mechanisms, it turns out that there is a strong need for further research in this area.

The SHARCS workshop, organized by ECRYPT's VAMPIRE Lab in February 2005, was the first open workshop that was devoted entirely to the challenging subject of making practical hardware crackers. In this report, we have reported on the most important contributions that have been presented during this workshop in Section 2.1.

In Chapter 3 we have surveyed hardware assisted factoring methods. All of them implement different steps of the GNFS. Two devices (Twinkle and Twirl) have received much attention after being proposed. Twinkle is an opto-electronic approach; LEDs are used to represent the progressions). Twirl is a purely electronic device. The Twinkle and the Twirl device both appear to lead to significant problems when implemented in practice. They are both designs that require to be implemented on a single large wafer. Using a 130nm process technology, TWIRL would cost 10 million US$ $\times$ year. Using the 90nm process technology, the cost of TWIRL would be 1.1 million US$ $\times$ year. It is however unclear how to create working ASICs of the desired size.

The SHARK device follows an approach that allows for a practical implementation on state-of-the-art technology. It has a modular architecture of small ASICs connected mainly via standard buses. It consists of 2300 identical isolated machines sieving in parallel. Each machine costs 70 000 US$. SHARK performs the sieving step of GNFS for a 1024-bit number within a year and costs around 200 million US$ (this includes 40 million US$ for the power consumption for one year).

In Chapter 4 we have surveyed methods for cracking DES. Although DES keys have been known to be too short since the first public DES cracker in 1998, a number of experiments have been conducted and are still running nowadays. In this deliverable we have underlined that the cost of attacks against the DES is now cheap, making it accessible to a larger number of institutions. In addition, we have discussed time-memory tradeoff attacks and underlined

that, for a similar cost to an exhaustive key search, one may perform the precomputation of such a tradeoff. All of the state-of-the-art attacks can be performed with standard equipment such as FPGAs and PCs. As a practical estimation, a US$ 12 000 machine could break DES in about 3 days, presently or in the near future.