

# Cryptanalysis with COPACOBANA

Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, *Member, IEEE*, and Andy Rupp

**Abstract**—Cryptanalysis of ciphers usually involves massive computations. The security parameters of cryptographic algorithms are commonly chosen so that attacks are infeasible with available computing resources. Thus, in the absence of mathematical breakthroughs to a cryptanalytical problem, a promising way for tackling the computations involved is to build special-purpose hardware exhibiting a (much) better performance-cost ratio than off-the-shelf computers. This contribution presents a variety of cryptanalytical applications utilizing the Cost-Optimized Parallel Code Breaker (COPACOBANA) machine, which is a high-performance low-cost cluster consisting of 120 field-programmable gate arrays (FPGAs). COPACOBANA appears to be the only such reconfigurable parallel FPGA machine optimized for code breaking tasks reported in the open literature. Depending on the actual algorithm, the parallel hardware architecture can outperform conventional computers by several orders of magnitude. In this work, we will focus on novel implementations of cryptanalytical algorithms, utilizing the impressive computational power of COPACOBANA. We describe various exhaustive key search attacks on symmetric ciphers and demonstrate an attack on a security mechanism employed in the electronic passport (e-passport). Furthermore, we describe time-memory trade-off techniques that can, e.g., be used for attacking the popular A5/1 algorithm used in GSM voice encryption. In addition, we introduce efficient implementations of more complex cryptanalysis on asymmetric cryptosystems, e.g., Elliptic Curve Cryptosystems (ECCs) and number cofactorization for RSA. Even though breaking RSA or elliptic curves with parameter lengths used in most practical applications is out of reach with COPACOBANA, our attacks on algorithms with artificially short bit lengths allow us to extrapolate more reliable security estimates for real-world bit lengths. This is particularly useful for deriving estimates about the longevity of asymmetric key lengths.

**Index Terms**—COPACOBANA, cryptanalysis, DES, A5/1, ECDLP, ECM, TMT0, e-passport.

## 1 INTRODUCTION

THE security of symmetric and asymmetric ciphers is usually determined by the size of their security parameters, particularly the key length. Hence, when designing a cryptosystem, these parameters need to be chosen according to the assumed computational capabilities of an attacker. Depending on the chosen security margin, many cryptosystems are potentially vulnerable to attacks when the attacker's computational power increases unexpectedly. In real life, the limiting factor of an attacker is often the financial resources. Thus, it is quite crucial from a cryptographic point of view to not only investigate the complexity of an attack but also study possibilities to lower the cost-performance ratio of attack hardware. For instance, a cost-performance improvement of an attack machine by a factor of 1,000 effectively reduces the key lengths of a symmetric cipher by roughly 10 bits (since  $1,000 \approx 2^{10}$ ). In this work, we make use of a special-purpose hardware system that can offer, depending on the application, a cost-performance ratio that is several orders of magnitude better than that of current PCs. The hardware architecture of this Cost-Optimized Parallel Code Breaker (COPACOBANA) has been introduced in [29]. In this contribution, we will describe further research on cryptanalytical applications over the last two years.

Cryptanalysis of modern cryptographic algorithms involves massive and parallel computations, usually requiring more than  $2^{40}$  operations. Many cryptanalytical schemes spend their computations in independent operations, which allows for a high degree of parallelism. Such parallel functionality can be realized by individual hardware blocks that operate simultaneously, improving the runtime of the overall computation by a perfect linear factor. At this point, it should be remarked that the high nonrecurring engineering costs for ASICs have put most projects for building special-purpose hardware for cryptanalysis out of reach for commercial or research institutions. However, with the recent advent of low-cost programmable ICs that host vast amounts of logic resources, special-purpose cryptanalytical machines have now become a possibility outside government agencies.

There are several approaches to building powerful computing clusters for cryptanalysis. For instance, distributed computing with loosely coupled processors connected via the Internet is a popular approach, e.g., demonstrated by the SETI@home project [46]. However, this has the disadvantage that the success strongly depends on the number of participating users. Hence, distributed computing usually results in an unpredictable runtime for an attack since the available computational power varies due to the dynamically changing number of contributors. A second intuitive approach could rely on utilizing supercomputers like IBM's BlueGene [32] or other commercial machines, e.g., from Cray or SGI. Supercomputers tend to provide sophisticated options for high-speed communication and large portions of distributed memory that are mostly not required for simple cryptanalytical number crunching. Unfortunately, the availability of these features increases

- The authors are with the Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Universitätsstr 150, 44780 Bochum, Germany. E-mail: {guneysu, tkasper, cpaar, arupp}@crypto.rub.de, novotnym@fel.cout.cz.

Manuscript received 1 Nov. 2007; revised 12 Mar. 2008; accepted 18 Mar. 2008; published online 1 May 2008.

Recommended for acceptance by W. Geiselmann, Ç. Koç, and R. Steinwandt. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2007-11-0565. Digital Object Identifier no. 10.1109/TC.2008.80.

the costs of these systems significantly, resulting in a nonoptimal cost-performance ratio of an attack on a cipher. With the improvements in field-programmable gate array (FPGA) technology, reconfigurable computing has emerged as a cost-effective alternative for certain supercomputer applications. FPGAs often get close to the computation power of custom hardware.

In this contribution, we will show how to use COPACOBANA for a variety of cryptanalytical applications. As already mentioned, the hardware is optimal for computational problems that are parallelizable onto independent nodes with low communication and memory requirements. COPACOBANA consists of up to 120 FPGA nodes that are connected by a shared bus, providing an aggregate bandwidth of 1.6 Gbps on the backplane of the machine. COPACOBANA is not equipped with dedicated memory modules but offers a limited number of RAM blocks inside each FPGA. Even though breaking modern ciphers like AES (128/192/256 bits of key), full-size RSA (1,024 bits or more) or Elliptic Curve Cryptosystems (ECCs; with 160 bits or more) is out of reach with COPACOBANA, we can use the machine to gather data for extrapolating attacks with realistic security parameters in terms of financial costs and attack time. Equally important, there are numerous legacy systems (and not-so-legacy systems such as the e-passport) that are still operating with key lengths that can be tackled with COPACOBANA.

Besides attacks on cryptographic primitives, we demonstrate attacking scenarios on unfavorably used cryptography in real-world applications. Current realizations of the basic access control (BAC) that shall prevent unauthorized access to the data stored on e-passports deploy symmetric cryptography based on SHA-1 and Triple Data Encryption Standard (DES). The corresponding encryption and authentication keys are generated from the data printed on the passport. As shown by several experts, the low entropy of the derived key allows for straightforward attacks with a relatively small complexity compared to an exhaustive key search attack on Triple DES. Using COPACOBANA, this kind of attack can be mounted almost in real time, i.e., the time needed for a person to pass an inspection system at the border control.

We have found several legacy systems based on a single DES cipher that are of practical relevance. We identified a class of crypto tokens generating One-Time Passwords (OTPs) according to the ANSI X9.9 standard, where the DES encryption is in use nowadays. Alarming, we are aware of online-banking systems in Europe and North and Central America that still distribute such tokens to users for authenticating their financial transactions.<sup>1</sup> Another cryptosystem employing the DES is Norton Diskreet, which has been a popular software encryption utility for files and hard disk drives based on a weak key derivation from a user-defined password. We also present how to employ Time-Memory Trade-Off (TMTO) and Time-Memory-Data Trade-Off (TMDTO) schemes on COPACOBANA. Both schemes use precomputed tables to improve the duration of exhaustive key search attacks.

1. Since we do not want to support hacking of bank accounts, we will not give further details here.

We show how COPACOBANA can support the precomputations for attacking the block cipher DES and the stream cipher A5/1 employed in the GSM system.

This manuscript is structured as follows: In the next section, we give a brief introduction about an FPGA-based special-purpose hardware architecture for breaking ciphers. Since the detailed concept has been fully presented in [29], we provide only a short overview here. In Section 3, we highlight selected applications for an exhaustive search on the key space of ciphers to recover the corresponding encryption key. An implementation of the DES on COPACOBANA impressively shows how DES can be broken with little effort in less than a week. This design is extended further for two real-world applications, i.e., extracting secrets from ANSI-X9.9-based crypto tokens and cracking the Norton Diskreet encryption software. Furthermore, we detail an attack aiming at identity theft with e-passports. After these straightforward brute-force attacks, we present more efficient ways of breaking multiple instances of a cipher with COPACOBANA based on TMTO and TMDTO in Section 4. After a brief introduction to TMTO, we describe implementations for TMTO attacks on the DES and the A5/1 cipher. Finally, we describe how COPACOBANA can attack or support attacks on asymmetric cryptosystems like RSA and ECC. Therefore, in Section 5, we present an efficient hardware implementation of the elliptic curve method (ECM) on COPACOBANA for factoring composite integers in parallel. Section 6 is dedicated to the Elliptic Curve Discrete Logarithm Problem (ECDLP), i.e., a widely used *one-way* function employed in ECC-based cryptosystems. For solving ECDLPs, we present a parallel implementation of Pollard's Rho (PR) algorithm on special-purpose hardware.

## 2 ARCHITECTURE OF COPACOBANA

The hardware architecture of COPACOBANA has been developed according to the following design criteria [29]: First, we assume that computationally costly operations are parallelizable. Second, parallel instances have only a very limited need to communicate with each other. Third, the demand for data transfers between host and nodes is low due to the fact that computations heavily dominate communication requirements. Ideally, (low-speed) communication between the hardware and a host computer is only required for initialization and the transfer of results. Hence, a single conventional (low-cost) PC should be sufficient to transfer the required data packets to and from the hardware, e.g., connected by a standardized interface. Fourth, all presented algorithms and their corresponding hardware nodes demand for very little local memory, which can be provided by the on-chip RAM modules of an FPGA.

Since the cryptanalytical applications demand for plenty of computing power, we installed a total of 120 FPGA devices on the COPACOBANA cluster. Building a system of comparable dimension with commercially available FPGA boards is certainly feasible but rather expensive. By stripping down the hardware functionality of COPACOBANA to the bare minimum and producing the hardware ourselves, we are able to achieve an optimal cost-performance ratio for code breaking. For a modular and maintainable architecture, we

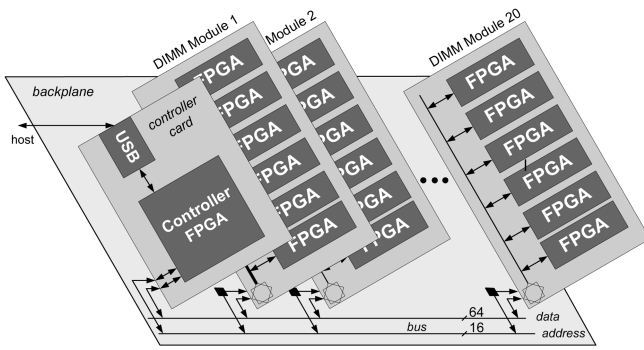


Fig. 1. Architecture of COPACOBANA.

designed small FPGA modules that can be dynamically plugged into a backplane. Each of these modules in DIMM form factor hosts six low-cost Xilinx Spartan-3 XC3S1000 FPGAs that are directly connected to a common 64-bit data bus onboard. The data bus of the module is interfaced to the global data bus on a backplane. While disconnected from the global bus, the FPGAs on the same module can communicate via the local 64-bit data bus. Additionally, control signals are run over a separate 16-bit address bus. Fig. 1 gives a detailed overview of the architecture of COPACOBANA. For simplicity, a single master bus was selected to avoid interrupt handling. Hence, if the communication scheduling of an application is unknown in advance, the bus master will need to poll the FPGAs.

The top-level entity of COPACOBANA is a host PC that is used to initialize and control the FPGAs, as well as for accumulation of results. Programming can be done simultaneously for all or a specific subset of FPGAs. Data transfer between FPGAs and a host PC is accomplished by a dedicated control interface. This controller has also been designed as a slot-in module so that COPACOBANA can be connected to a computer either via a USB or Ethernet controller card. A software library on the host PC provides low-level functions that allow for device programming, addressing individual FPGAs, and storing and reading FPGA-specific application data. With this approach, we can easily attach more than one COPACOBANA device to a single host PC.

### 3 EXHAUSTIVE KEY SEARCH SCENARIOS

The impracticability of an exhaustive key search, i.e., testing each key of the corresponding key space, is a precondition for the security of symmetric ciphers. The cost of such an attack is calculated based on the available technology and expected future developments. Usually, the key size is chosen such that it allows for a fast and efficient implementation of the cryptosystem on the one hand but makes such brute-force attacks impracticable on the other hand.

#### 3.1 Exhaustive Key Search on DES

The DES with a 56-bit key size was chosen as the first commercial cryptographic standard by NIST in 1977 [35]. A key size of 56 bits was considered to be a good choice considering the huge development costs for computing power in the late 1970s, which made a search over all the

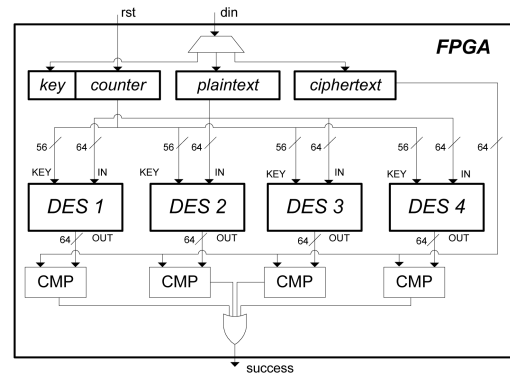


Fig. 2. Architecture for exhaustive key search with four DES key search units.

possible  $2^{56}$  keys appear impractical. There have been a lot of feasibility studies on the possible use of parallel hardware and distributed computing for breaking DES. The first estimates were proposed by Diffie and Hellman in 1977 [10] for a brute-force machine that could find the key within a day at a cost of US\$ 20 million. In 1998, the Electronic Frontier Foundation (EFF) built a DES hardware cracker called *Deep Crack*, which could perform an exhaustive key search within 56 hours [11]. Their DES cracker consisted of 1,536 custom-designed ASIC chips at a cost of material of around US\$ 250,000 and could search 88 billion keys per second. Though DES is known to be broken and obsolete today, the costs for building a machine like *Deep Crack* can still be considered impractical for smaller organizations. COPACOBANA is a more practical and affordable approach as a programmable off-the-shelf hardware cracker.

Since DES has been designed to be extremely efficient in terms of area and speed in hardware, an FPGA implementation of DES can be orders of magnitude faster than an implementation on a conventional PC at much lower costs. This allows a hardware-based engine for a DES key search to be much faster and efficient compared to a software-based approach.

Our core component is an improved version of the DES engine of the Université Catholique de Louvain's Crypto Group [43] based on 21 pipeline steps. Our design can test one key per clock cycle and the pipelined architecture is adjusted such that the critical path is as small as possible. On the COPACOBANA, we can fit four of such DES engines inside a single FPGA, which allows for sharing plaintext-ciphertext input pairs and the key space, as shown in Fig. 2.

Since our first implementation, as presented in [29], we have been able to tweak our design for increased performance by the use of additional pipelined comparators and improved control logic. Now, we can operate each of the FPGAs at an increased clock rate of 136 MHz so that a gain in performance by 36 percent is achieved, compared to that in [29]. Consequently, an amount of  $2^{42}$  keys can be checked in  $2^{40} \times 7.35$  ns by a single FPGA, which is approximately 135 minutes. Since COPACOBANA hosts 120 of these low-cost FPGAs, the key search machine can check  $4 \times 120 = 480$  keys every 7.35 ns, i.e., 65.28 billion keys per second. To find the correct key, COPACOBANA

has to search through an average of  $2^{55}$  different keys. Thus, it can find the right key in approximately  $T = 6.4$  days on the average. Of course, more than one COPACOBANA can be attached to a single host and the key space shared so that the search time is reduced to  $\frac{T}{n}$ , where  $n$  denotes the number of machines.

To compare the cost-performance ratio of COPACOBANA with respect to a software-based approach, let us relate both architectures according to a constrained budget. With an expense of € 10,000 required for the material of a COPACOBANA, we can afford about 50 low-cost PCs (Pentium 4 at 3 GHz including necessary peripherals) for € 200 each in equal measure. A standard software implementation of DES can compute about 2 million DES encryptions per second on such a PC. Hence, with the fixed investment of € 10,000, we here yield a throughput of 100 million DES keys per second with the PC cluster. Compared to the 65.28 billion DES keys searched by a single COPACOBANA per second, we can outperform the PC cluster in this case by a factor of more than 650. Regarding a power consideration, we measured a fully equipped COPACOBANA running a DES key search to consume less than 600 W. Related to this, we assume a single Pentium 4-based computer to require 150 W on the average. Hence, comparing the power consumption of the entire key search on a COPACOBANA and the PC cluster, a worst case key search on COPACOBANA will take 184 kWh, whereas the PC cluster consumes the immense amount of about 1.5 GWh during runtime.

### 3.2 Extracting Secrets from ANSI-X9.9-Based Crypto Tokens

In a real-world scenario, we have mounted an attack on cryptographic tokens that are used for user authentication and identification according to FIPS 113 or ANSI X9.9. This technique is based on OTPs generated using the DES algorithm and is still used in many security-relevant applications.<sup>2</sup> We assume that OTP tokens have a securely integrated static key inside and do not rely on time- or event-dependent methods for computing the passwords (e.g., contrary to RSA SecurID tokens). In combination with a challenge-response protocol, a decimal-digit challenge is manually entered into the token via an integrated keypad. The token in turn computes the corresponding response according to the ANSI X9.9 standard. Tokens implementing this standardized authentication scheme (incorporating ANSI 3.92 DES encryption) often have a fixed-size LCD, allowing displaying eight decimal digits for input and output. Fig. 3 graphically shows how the response is generated by the token according to a given challenge. The mapping  $\mu$  is used to convert the hexadecimal digits from the output to decimal representation to be displayed on the LCD. We can prove that with at least two pairs of challenge-response data, we can perform an exhaustive key search on the DES key space implementing the specific features of ANSI X9.9 authentication, giving only 16 key candidates on the average.

2. We are aware of online-banking systems in some places of the world still relying on ANSI-X9.9-based tokens for authorization of financial transactions. We prefer not to give any details at this point.

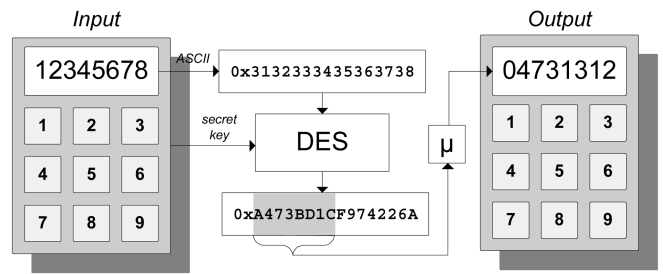


Fig. 3. Principle of the response derivation of a DES-based crypto token.

Assuming the DES encryption function to behave like a pseudorandom function with appropriate statistical properties, the 32 most significant bits of the DES output  $c$  can be regarded as eight hexadecimal digits uniformly distributed over  $H = \{0, \dots, 9, A, \dots, F\}$ . The digits are converted to  $F = \{0, \dots, 9\}$ , where  $T = \{0, \dots, 5\}$  are doubly assigned. Hence, we know that  $\Delta = F \setminus T = \{6, \dots, 9\}$  are four fixed points that directly correspond to the output digits of  $c$ , yielding four bits of key information. The six remaining decimal digits  $\Omega = F \cap T$  can have two potential origins, resulting in a variance of one bit. We can assume that the probability for an arbitrary digit  $i$  of  $c$  being in  $\Delta$  is  $\Pr[i \in \Delta] = 1/4$ , expecting two out of eight hexadecimal digits of  $c$  to be fixed points. When averaged, this leads to knowledge of  $R = 2 \cdot 4 + 6 \cdot 3 = 26$  bits of DES key material. With two plaintext-ciphertext pairs, we then have 52 bits of key information on the average, resulting in 16 possible key candidates. With this small number of potential solutions, the attacker can attempt to guess the right solution by trial and error. However, in the case that three challenge-response pairs are given, we are able to exactly determine the key in a single exhaustive key search. We have implemented the corresponding FPGA architecture for an extended exhaustive key search according to the input of two challenge-response pairs. Our design is again based on the exhaustive key search architecture as shown in Section 3.1 with additional logic for two pairs of input and the final ciphertext conversion by  $\mu$ . After placing and routing, the device usage of 8,729 flip-flops (56 percent of Spartan-3 XC3S1000 device) and 12,813 lookup tables (LUTs, 83 percent of Spartan-3 XC3S1000) running at 120 MHz has been reported by Xilinx ISE 9.1. Therefore, a fully equipped COPACOBANA with 120 FPGAs can compute 57.6 billion outputs of ANSI X9.9 authenticators per second so that a successful key search will require 7.2 days. In other words, when attempting to spoil bank accounts, an investment of € 1 million in COPACOBANA systems can break such an account in less than 2 hours given three challenge-response pairs acquired, for example, by phishing attacks.

### 3.3 Cracking Norton Diskreet

In the 1990s, Norton Diskreet, a part of the well-known Norton Utilities package, was a very popular encryption tool. Diskreet can be used to encrypt single files, as well as to create and manage encrypted virtual disks. The tool provides two encryption algorithms that one can choose from: a (cryptographically very weak) proprietary algorithm and the DES in cipher block chaining (CBC) mode.

Parts of the internals and flaws of Diskreet that we consider in the following have also been reported in newsgroup postings by Gutmann [38] and Kocher [39].

### 3.3.1 DES Key Generation

To encrypt a file or virtual disk, Diskreet asks for a password with a minimal length of 6 bytes and a maximal length of 40 bytes. From this password, the 56-bit DES key is generated. The password-to-key mapping works as follows: First, leading whitespace characters are removed before the password is converted to uppercase characters, which are divided into chunks of 8 bytes. Then, all 8-byte blocks are subsequently XORed with each other, and the resulting sum is used as the DES key. Obviously, this method of key generation is unfavorable since the password-to-key mapping is not chaotic at all. More precisely, depending on the kind of characters of a password, we obtain the following subspaces of the DES key space:

1. *Key space  $\alpha$* . Let us assume that all characters of a password are from the set  $\{A, \dots, Z, @, [, \backslash, ^, -, _\}$ . This is the set of all ASCII characters in the range 64-95. Thus, the binary representation of each password character has the form 010xxxxx. Hence, due to the XOR operation, each byte of the resulting DES key can either have the form 010xxxxx or 000xxxxx. Whether a key byte corresponds to the first or the second form depends on the position of the byte and the length of the password. It is easy to see that the password length modulo 16 uniquely determines which byte of a key matches which pattern, i.e., for a particular password length mod 16, there is exactly one key pattern. Since the least significant bit of each key byte is a parity bit,  $\alpha$  contains a total of  $16 \cdot 2^{32} = 2^{36}$  DES keys. If the password length is known a priori, the effective key length is reduced to 32 bits.
2. *Key space  $\beta$* . Let us assume that the password only consists of 7-bit ASCII characters. Then, each key byte matches the pattern 0xxxxxxx, where the least significant bit can be ignored again. Hence,  $\beta$  contains  $2^{48}$  keys.
3. *Key space  $\gamma$* . If we consider passwords consisting of arbitrary 8-bit ASCII characters, we obtain the whole DES key space, which contains  $2^{56}$  different keys. We denote this key space by  $\gamma$ .

### 3.3.2 Password Check

Before performing a decryption, Diskreet first checks whether the correct password has been entered. To enable this kind of verification process, Diskreet performs the following additional steps prior to the actual encryption of user data: it puts the XOR sum of the DES key  $K$  destined for encryption and an 8-byte mask  $M$  in the header of the file that is used for storing the encrypted data. Then, it encrypts the part of the header that contains  $K \oplus M$  with  $K$  using DES in CBC mode. We denote the corresponding 8-byte ciphertext by  $C$ . After that, the mask  $M$  is stored in the plaintext part of the header.

TABLE 1  
Breaking Norton Diskreet with COPACOBANA

Key space	Remark	DES decryptions (on average)	Runtime
$\alpha$	Known pwd length	$2^{31}$	32.8 $\mu s$
$\alpha$	Unknown pwd length	$2^{35}$	0.53 s
$\beta$	7-bit ASCII	$2^{47}$	35.9 m
$\gamma$	8-bit ASCII	$2^{55}$	6.39 d

Hence, in order to verify the correctness of a key  $K'$ , one simply needs to test the following equality:<sup>3</sup>

$$K' = \text{DES}_{K'}^{-1}(C) \oplus M \oplus C', \quad (1)$$

where  $C'$  denotes the 8-byte block of ciphertext that is located just before  $C$ . In the case of Diskreet's password check,  $K'$  is generated from the entered password by applying the mapping described in Section 3.3.1.

### 3.3.3 Key Search Using COPACOBANA

An exhaustive key search for Norton Diskreet can easily be performed by a slight modification of the circuit depicted in Fig. 2. Due to marginal changes, we still assume four DES cores on a single FPGA. The register for the plaintext contains the constant  $C$  from (1); the register for the ciphertext contains the (constant) value of  $M \oplus C'$ , which can be computed in advance on the host PC. Instead of simply comparing the actual result of a decryption with the value of the ciphertext register, we now have to compute an XOR of the register's content with the actual key and compare the result with the corresponding output of the DES core. Hence, an additional XOR operation is required.

Depending on the actual key space ( $\alpha$ ,  $\beta$ , or  $\gamma$ ), the counter output has to be connected to the four DES cores in different ways. In the case of key space  $\gamma$ , the key search can be adopted from Section 3.1. The overall runtime of an average key search of  $2^{56}$  keys does not change. With key space  $\beta$ , the fixed part of the key can be reduced by 7 bits, reducing the time of an average search of  $2^{48}$  keys by a factor of  $2^8$  compared to key space  $\gamma$ .

Table 1 summarizes the number of DES operations and absolute timings required to break Diskreet with COPACOBANA.

## 3.4 Identity Theft with Electronic Passports

The e-passport, as specified by the International Civil Aviation Organization (ICAO), is deployed in many countries all over the world. The security and privacy threats have been widely discussed (e.g., [26], [28], [20], and [25]) and have provoked public debates. In this section, we give details about our hardware implementation of the security mechanism used for the access control of the e-passport and present practical figures for an exhaustive key search on the COPACOBANA.

A chip embedded in the machine-readable travel document (MRTD) contains private data as text, such as name, date of birth, and gender, as well as biometrics [34]. A digital facial photograph and, in some countries,

3. Note that the equality check ignores the least significant bit of each byte.

additionally fingerprints or an iris scan of the passport holder can be accessed via a contactless interface based on the ISO 14443 [24] standard. The wireless communication constitutes new opportunities for attackers, such as relay attacks [27] or eavesdropping from a range of several meters, as investigated in [17], [42], and [13]. To prevent unauthorized access to the information transferred via the radio frequency (RF) interface, some countries, among them Germany and The Netherlands, employ the so-called BAC. The BAC is meant to secure the interchanged data, i.e., establish a confidential channel, by employing symmetric cryptography.

The secret keys needed for carrying out the BAC are stored in the embedded IC and can also be derived from a machine-readable zone (MRZ) that is printed on the paper document. Hence, before an e-passport reader can communicate with a passport using BAC, e.g., at the border control, it has to optically scan the MRZ. With the MRZ information, it can generate the secret keys  $k_{ENC}$  and  $k_{MAC}$  for the encryption and generation of a Message Authentication Code (MAC), respectively. Then, the mutual three-pass authentication according to the BAC protocol [21] is carried out as follows:

At the beginning of the BAC, the e-passport generates a random number  $RND_{Epass}$  and sends it to the reader. The reader concatenates  $RND_{Epass}$  with more random bits  $RND_{Reader}$  and encrypts the result with  $k_{ENC}$  to obtain  $E_{Reader} = ENC_{k_{ENC}}(RND_{Epass} || RND_{Reader})$ . In addition, a MAC  $M_{Reader} = MAC_{k_{MAC}}(E_{Reader})$  is appended before  $E_{Reader} || M_{Reader}$  is returned to the passport.<sup>4</sup> After decrypting the data, the MRTD verifies the received  $RND_{Epass}$  against the original value and so assures that the reader possesses the correct secret key  $k_{ENC}$ . Finally, the e-passport proves its knowledge of  $k_{ENC}$  to the reader in a similar fashion.  $RND_{Epass}$  is again concatenated with a part of  $RND_{Reader}$  plus some more random bits before being encrypted to  $E_{Epass} = ENC_{k_{ENC}}(RND_{Epass} || \dots)$ . After generation of the MAC  $M_{Epass} = MAC_{k_{MAC}}(E_{Epass})$ , the concatenation of the ciphertext with the MAC, i.e.,  $E_{Epass} || M_{Epass}$ , is again broadcasted via the RF link. The random bits generated additionally by the reader and the passport are used for the derivation of a session key that is used only once for the encryption of the subsequent communication. For a fully detailed description of the BAC and further security mechanisms integrated in the MRTD, refer to [21], [23], [22], and [12].

For our attack scenario, adapted from [6], we assume that a device for eavesdropping of the RF field can be mounted near an e-passport inspection system such that all bits transmitted via the air channel can be captured and stored in a database. An attacker thus has two options for gaining the couple of plaintext and ciphertext needed for the key search on the MRZ.

The first option targets  $k_{ENC}$ . The plaintext for this case is the 64-bit number  $RND_{Epass}$  transmitted by the passport at the beginning of the BAC. The corresponding ciphertext  $ENC_{k_{ENC}}(RND_{Epass})$  is part of the last message  $E_{Epass} || M_{Epass}$  that is sent during the BAC. The encryption function  $ENC_{k_{ENC}}(\cdot)$  is Triple DES in CBC mode, with the initialization vector being publicly known [21]. Hence, the most significant 8 bytes  $msb_8(E_{Epass} || M_{Epass})$  can be de-

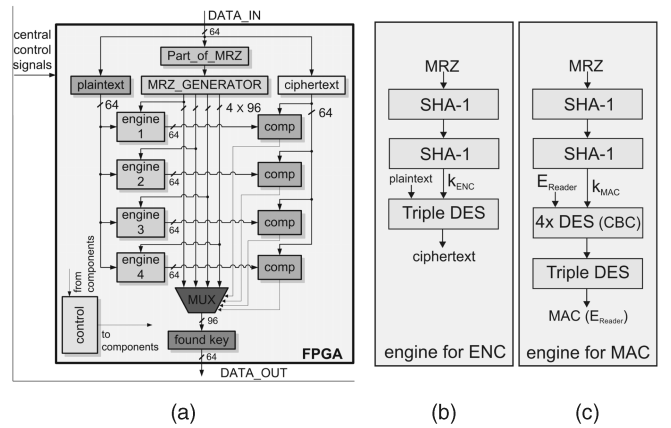


Fig. 4. (a) Content of one FPGA. (b) Details about the engines for attacking  $k_{ENC}$ . (c) Details about the engines for attacking  $k_{MAC}$ .

rypted with varying keys for comparison with  $RND_{Epass}$ , without knowledge of the remainder of the datagram. In case of a match,  $k_{ENC}$  and, thereby, the related MRZ are found.

In practice, the bitstream transmitted by the e-passport is much more difficult to eavesdrop than the request of the reader [14]. If monitoring of the data transmitted by the reader is the only option, an attacker can still gain the MRZ and, hence, the secret keys of an e-passport following our second approach. This time, the MAC key  $k_{MAC}$  is targeted, and intercepting only one message of the reader, i.e.,  $E_{Reader} || M_{Reader}$ , is sufficient for identifying a particular e-passport. Obviously, here,  $E_{Reader}$  is the plaintext, and its MAC  $M_{Reader}$  will match with  $MAC_{k_{MAC}}(E_{Reader})$  in case of a correct key guess for  $k_{MAC}$ .

The keys  $k_{MAC}$  and  $k_{ENC}$  are derived from the MRZ information according to

$$k = msb_{16}(\text{SHA-1}(msb_{16}(\text{SHA-1}(\text{MRZ})) || C)).$$

After the first execution of SHA-1 [37], the result is concatenated with a constant  $C$ , which is either  $C = 0x00000001$  for  $k_{ENC}$  or  $C = 0x00000002$  for  $k_{MAC}$ . Thus, to obtain one key from the MRZ information, two subsequent rounds of SHA-1 have to be executed for both of the above approaches.

As the COPACOBANA does not provide enough memory for storing precomputations, SHA-1 is the most time consuming part of our implementation. Its critical path sets the upper limit for the clock frequency, and 80 clock cycles are needed for obtaining one hashed output. As two subsequent SHA-1 need to be executed for one key, we provide each FPGA with two pipelined SHA-1 units.

It turns out that when implementing the residual parts of the proposed attacks, the additional area occupied by the second SHA-1 does not restrict the performance. The first option for the attack involving  $ENC_{k_{ENC}}(\cdot)$  demands for a Triple DES to be executed after hashing the key, as depicted in Fig. 4b. This is achieved by letting run only one DES round for 48 times (16 times for each single DES), instead of a fully parallel DES consuming much more logical gates. Still, the FPGA is idle for  $80 - 48 = 32$  clock cycles until the next output of the SHA-1 is determined so that the most efficient time-area trade-off for our design is found.

4.  $MAC_{k_{MAC}}(\cdot)$  denotes the cryptographic checksum according to ISO/IEC 9797-1 MAC Algorithm 3, as detailed in the annex of [21].

For the second proposed attack targeting the MAC, four more DES rounds have to be executed in addition to the Triple DES, as illustrated in Fig. 4c. Hence,  $7 \times 16 = 112$  clock cycles would be needed after each SHA-1 computation if only one round of DES was implemented. This would have a bad impact on the overall performance, as  $112 - 80 = 32$  clock cycles of idle time would occur each time a new hash value is delivered. Instead, our solution implements a second round of DES on each FPGA for which sufficient area is available after some optimizations of the control logic. Now, the postprocessing after the SHA-1 takes 56 clock cycles and is thus  $80 - 56 = 24$  clock cycles faster than the SHA-1. Indeed, targeting  $k_{MAC}$  is as costly, with respect to execution time, as aiming at  $k_{ENC}$ , so that the following discussion concerning the first approach is equally valid for the second attack option.

The attack implementation on an FPGA is shown in Fig. 4a. Each FPGA stores the same pair of plaintext and ciphertext in the corresponding registers. Since 120 FPGAs are available on COPACOBANA, the key space is split into appropriate subspaces. These subspaces are allocated to the FPGAs by means of the Part\_of\_MRZ register, which contains a portion of the MRZ that is fixed for each particular FPGA. An MRZ generator produces all remaining combinations of the MRZ and supplies them to four engines that process the plaintext in parallel, as detailed above. The delivered ciphertexts are compared to the correct one stored in the ciphertext register. In case of a match, the sought-after MRZ is returned to the data bus.

The MRZ generator is a very important part of our design, as it minimizes the communication via the data buses, which is a well-known bottleneck of the architecture of the COPACOBANA. After initialization of the registers each FPGA runs independently, and no more information interchange is needed until a key is found. Furthermore, the MRZ generator allows for a flexible distribution of the key space and adapting the key search depending on the knowledge about the passport holder and the issuing system of the e-passport.

The latter property is extremely important for breaking BAC keys, as the entropy of the MRZ can be considerably reduced [31], [42] with an increasing knowledge of the adversary. For example, the date of birth of the passport holder, which can be known or guessed, is part of the MRZ. The passport expiry date is another portion of the MRZ whose entropy is limited due to the passport issuing schemes of the respective countries. Furthermore, the expiry date is correlated with the serially increased passport number, being the third and last component that the MRZ consists of. Eventually, the entropy can be reduced to as low as  $\approx 2^{33}$  for realistic scenarios based on the BAC realizations of The Netherlands and Germany, which are the focus of our implementation.

As stated earlier, the time critical component is SHA-1, determining the maximum clock frequency of  $f_{clk} = 40$  MHz and requiring 80 clock cycles for one key candidate. The processing of one key thus requires  $80 \times 25 \text{ ns} = 2 \mu\text{s}$ . As there are 120 FPGAs running in parallel, each possessing four encryption engines,  $4 \times 120 = 480$  keys are tested every  $2 \mu\text{s}$ , resulting in a throughput of  $2^{27.84} \approx 240$  million

keys per second. On the average, testing of  $2^{33}$  keys reveals the correct candidate in  $\frac{2^{32}}{2^{27.84}} \approx 18$  seconds, which can be regarded as real time, compared to the duration of one inspection at the border control.

Our implementation for breaking BAC keys on the COPACOBANA shows that the practical realization of the BAC in The Netherlands and in Germany should be regarded critically, as an adversary can gather private information about passport holders, including biometrics, from a distance. This is possible due to the low entropy of the MRZ. Our results also show that if the full entropy had been used, an attack would be practically infeasible even with the special-purpose hardware currently at hand. Still, the key search performance could be significantly increased if fast onboard RAM for precomputations could be made available in future realizations of cryptographic key search machines.

#### 4 TIME-MEMORY TRADEOFF ATTACKS

The inversion of (one-way) functions is a common problem frequently appearing in cryptanalysis: Let  $g : X \rightarrow Y$  be a (one-way) function with a domain  $X$  of size  $|X| = N$ . Given an image  $y \in Y$ , the challenge is to find a preimage of  $y$ , i.e., some element  $x \in X$  such that  $g(x) = y$ . Instances of this problem appear in the cryptanalysis of block and stream ciphers. In the case of a block cipher  $E$ , one is typically given a *fixed* known plaintext  $P$  and tries to invert the bijective function:

$$\begin{aligned} g_P : X &\rightarrow Y, \\ x &\mapsto E_x(P), \end{aligned}$$

mapping keys  $x$  to ciphertexts  $y$  of  $P$ , for a given ciphertext  $g_P(x')$ . In the case of a stream cipher, the domain  $X$  of the function  $g$  that one tries to invert is the set of all possible internal states of the cipher. The function  $g$  maps an internal state to the first  $\log_2(|X|)$  output bits of the cipher produced from this state. Typically, one is given several of these output strings  $y_1, \dots, y_D$ , and it is already sufficient to find a preimage for *one* of them.

By using a cryptanalytic TMTO method, one tries to find a compromise between the two well-known extreme approaches, i.e., performing exhaustive searches and precomputing exhaustive tables, to solve this general problem. A TMTO offers a way to reasonably reduce the actual search complexity (by doing some kind of precomputation) while keeping the amount of precomputed data reasonably low, whereas “reasonably” has to be defined more precisely. It depends on the concrete attack scenario (e.g., real-time attack), the function  $g$ , and the available resources for the precomputation and online (search) phase.

Existing TMTO methods [8], [19], [36] share the natural property that in order to achieve a significant success rate, much precomputation effort is required. Since performing this task on PCs is usually way too costly or time consuming, cheap special-purpose hardware with massive computational power, like COPACOBANA, is demanded. In [45], an FPGA design for an attack on a 40-bit DES variant using Rivest’s TMTO method [8] was proposed. In [33], a hardware architecture for Unix password cracking

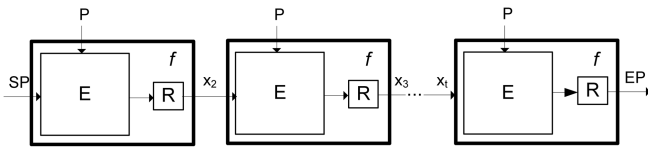


Fig. 5. Chain generation according to Hellman's TMTO.

based on Oechslin's method [36] was presented. However, to the best of our knowledge, nobody has done a complete TMTO precomputation for *full* 56-bit DES so far, let alone ciphers of greater relevance like A5/1.

In Section 4.1, we give a brief overview of cryptanalytic TMTO methods, which is followed by the design and implementation of a TMTO attack on DES presented in Section 4.2. Then, we shortly describe the idea of TMDTOs in Section 4.3 and present a TMDTO attack implementation targeting A5/1 in Section 4.4.

#### 4.1 Time-Memory Tradeoff Methods in Cryptanalysis

In this section, we sketch Hellman's original TMTO method, as well as the variants proposed by Rivest and Oechslin. For concreteness, the methods are considered in the case of a block cipher  $E$  given a fixed known plaintext  $P$ , i.e., we want to invert the one-way function  $g_P(x) = E_x(P)$ .

##### 4.1.1 Hellman's Original Approach

In Hellman's TMTO attack, published in 1980 [19], one tries to precompute all possible key-ciphertext pairs in advance by encrypting  $P$  with all  $N$  possible keys. However, to reduce memory requirements, these pairs are organized in several *chains* of fixed length. The chains are generated deterministically and are uniquely identified by their respective start and end points. In this way, it suffices to save its start and end point to restore a chain later on. In the online phase of the attack, one then simply needs to identify and reconstruct the right chain containing the given ciphertext to get the wanted key. The details of the two phases are described in the following.

**Precomputation phase.** In this phase, first,  $m$  different keys are chosen to serve as start points  $SP$  of the chains. To generate a chain, one first computes  $E_{SP}(P)$ , resulting in some ciphertext  $C$  (see Fig. 5). In order to continue the chain,  $C$  is used to generate a new key. To this end, a so-called *reduction and rerandomization function*  $R$  is applied reducing the bit length of  $C$  to the bit length of a key for the cipher  $E$  (if necessary) and performing a rerandomization of the output. By means of  $R$ , we can continue the chain by computing  $R(E_{SP}(P)) = x_2$ , using the resulting key  $x_2$  to compute  $R(E_{x_2}(P)) = x_3$  and so on. The composition of  $E$  and  $R$  is called *step function*  $f$ . After  $t$  applications of  $f$ , the chain computation stops, and we take the last output as the end point  $EP$  of the chain. The pair  $(SP, EP)$  is stored in a table sorted by the end points. The number of distinct keys contained in a table divided by  $N$  is called the *coverage* of a table. Unfortunately, the occurrence of a key in a table is not necessarily unique because there is a chance that two chains collide and merge or that a chain runs in a loop. This is due to the noninjective function  $R$  mapping the space of ciphertexts to the space of keys (which is often smaller, e.g., in the case of

DES). Each merge or loop reduces the fraction of distinct keys contained in a table and, thus, the coverage (if  $m$  is fixed). Since the probability of merges increases with the size of a table, at a certain point, we cannot significantly improve the coverage by simply adding more and more chains. Hellman calculated that this point is somewhere near  $N^{\frac{2}{3}}$  for a single table. To cope with this problem, he suggested to generate multiple tables, each associated with a different reduction function. In this way, even if two chains from different tables collide, they will not merge because different functions are applied to the shared value in the next step.

**Online phase.** In the online phase, a ciphertext  $C'$  is given, which is assumed to be the result of the encryption of  $P$  using some key  $k$ . We try to retrieve  $k$  from the precomputed tables in the following way: to find out if  $k$  is covered by a specific table, we compute a chain up to a length of  $t$  starting with  $R(C')$  and compare the intermediate points with the end points in the table. More precisely, we first check if  $R(C')$  is contained. If not, we compute  $f(R(C'))$  and look for a match; then, we do this for  $f(f(R(C')))$  and so on. If a match occurs after the  $i$ th application of  $f$  for a pair  $(SP, EP)$ , then  $f^{t-i-1}(SP) = x_{t-i}$  is a key candidate. This candidate needs to be checked, by verifying  $E_{x_{t-i}}(P) = C'$ , and if it is valid, the online phase ends. If it is not valid, a *false alarm* has occurred, and the procedure continues while the chain has a length smaller than  $t + 1$ . If no valid key is found in this table, we repeat the same procedure for another table (and, thus, another  $R$  and  $f$ ).

##### 4.1.2 Variants of Hellman's Approach

**Distinguished points (DPs).** In practice, the time required to complete the online phase of Hellman's TMTO is dominated by the high number of table accesses. Random accesses to the disk can be many orders of magnitude slower than the evaluation of  $f$ . The DP method, introduced by Rivest [8] in 1982, addresses this problem. A DP is a key that fulfills a certain simple criterion (e.g., the first 20 bits are 0), which is usually given as a mask of length  $d$ . Rivest's idea was to admit only DPs as end points of a chain. For the precomputation phase, this means that a chain is computed until a DP or a maximal chain length  $t_{max} + 1$  is reached. Only chains of length at most  $t_{max} + 1$  ending in a DP are stored. Using DPs, merging and looping chains can also be detected and are discarded. In the online phase, the table does not need to be accessed after every application of  $f$  but only for the first occurring DP. If we have no match for this DP, we can proceed with the next table.

**Rainbow tables.** Rainbow tables were introduced by Oechslin [36] in 2003. He suggested not to use the same  $R$  when generating a chain for a single table but a (fixed) sequence  $R_1, \dots, R_t$  of different reduction functions. More precisely, due to the different reduction functions, we get  $t$  different step functions  $f_1, \dots, f_t$  that are applied one after another in order to create a chain of length  $t + 1$ . The advantage of this approach is that the effect of chain collisions is reduced: while in a Hellman table, the collision of two chains inevitably leads to a merge of these chains, in a rainbow table, a merge only happens if the shared value appears at the *same* position in both chains. Otherwise, they share only this single value. Thus, a merge of two chains in a rainbow table is not likely to occur. Furthermore, loops are



TABLE 2  
Empirical TMTO Parameters for Optimal Performance

Method	Chain Length	# SPs	# Tables	# Bits p.E.
Hellman	$t = 2^{19.2}$	$2^{16.7}$	$2^{21}$	73
Rainbow	$t = 2^{19.5}$	$2^{35}$	5	91
DP	$t_{min} = 2^{18}$ , $t_{max} = 2^{20}$ , $d = 19$	$2^{18}$	$2^{21}$	55

completely prevented. Hence, regarding a space-efficient coverage, these characteristics allow us to put much more chains into a rainbow table than into a Hellman table. This in turn significantly reduces the total number of tables needed in order to achieve a certain coverage. Since fewer rainbow tables must be searched in the online phase (which is, however, a bit more complex), a lower number of calculations and table accesses are required compared to Hellman’s method. To look up a key in a rainbow table, we first compute  $R_t(C')$  and compare it to the end points; then, we do this for  $f_t(R_{t-1}(C'))$ ,  $f_t(f_{t-1}(R_{t-2}(C')))$ , etc. Moreover, compared to the DP method, the number of false alarms and the induced extra work are reduced.

## 4.2 A TMTO Attack on DES

In this section, we will employ COPACOBANA for accelerating the precomputation and online phase of a TMTO attack on DES. In such a scenario, primarily, the hardware limitations of COPACOBANA with respect to communication demands need to be taken into account. Since COPACOBANA does not allow the installation of directly attached storage, all TMTO tables must be managed by the connected host PC. The current USB interface between COPACOBANA and the host PC provides a communication bit rate of  $24 \cdot 10^6 \approx 2^{24.5}$  bits per second.<sup>5</sup> Compared to the number of possible DES encryptions per second, the bottleneck of the COPACOBANA is the data throughput for transferring  $(SP, EP)$  tuples from the FPGAs to the host. To address the constraint of limited bandwidth, we have determined a minimum rate of  $2^{11.4} \cdot b$  computations to be run in sequence until a data transfer can be initiated, where  $b$  denotes the aggregate bitlength of a tuple  $(SP, EP)$ . For practical reasons, we have limited the disk space for the TMTO tables to a maximum of 2 Tbytes and the required success rate to 80 percent. Based on experiments, we determined the parameters for the chain length, the number of tables, and the start points satisfying the given constraints. These parameters are shown in Table 2.

To reduce data transfers to a bare minimum, we use the first  $m$  integers as start points  $SP$  and assign fixed subintervals of  $[0, m]$  to each FPGA. In this way, each  $SP$  can be stored with only  $\log_2(m)$  bits. Optionally, the host PC can even track the sequence of start points for each individual FPGA so that data transfers of  $SP$ s can be omitted completely. Then, only the end points  $EP$  must be transmitted to the host PC and matched with the corresponding  $SP$  software counter.

5. Please note that we are currently working on a Gigabit Ethernet solution so that subsequent calculations based on the limited bandwidth may be subject to change.

TABLE 3  
Expected Runtimes and Memory Requirements

Method	SR	DU	PT (COPA)	OT
Hellman	0.80	1897 GB	24 days	$2^{40.2}$ TA + $2^{40.2}$ C
DP	0.80	1690 GB	95 days	$2^{21}$ TA + $2^{39.7}$ C
Rainbow	0.80	1820 GB	23 days	$2^{21.8}$ TA + $2^{40.3}$ C

For the DP method, we introduce a minimum chain length  $t_{min}$  to ensure that the generated data traffic from tuples  $(SP, EP)$  always complies with the available bandwidth on the COPACOBANA. More precisely, each DP chain leading to a total chain length of less than  $t_{min} + 1$  is discarded and not transferred to the host.<sup>6</sup> The storage of end points for the DP method can be limited to the remaining  $56 - d$  bits not covered by the DP criterion.

Table 3 presents our worst case expectations concerning the success rate (SR), the disk usage (DU), the duration of the precomputation phase (PT) on COPACOBANA, and the number of table accesses (TA) and calculations (C) during the online phase (OT). Note that these figures for use with COPACOBANA are based on estimations given in [19], [36], and [45] (false alarms are neglected) and the given constraints mentioned above. Note further that for this initial extrapolation, we have used the implementation of our exhaustive key search unit presented in Section 3.1. According to our findings, precomputations for the DP method on a single COPACOBANA take roughly four times longer compared to Hellman’s and Oechslin’s method based on the given constraints. In contrast, the subsequent online attack has the lowest complexity for the DP method. Considering a TMTO scenario involving COPACOBANA for precomputation only (implying that the online attack is performed by a PC), the rainbow table method can be assumed to provide the best performance. When using COPACOBANA as well for precomputation *and* online phase, there is a strong indicator to select DPs as the method of choice: for the DP method, we can assume the frequency of table accesses to follow a uniform distribution; hence, we expect balanced bandwidth requirements over time. With respect to the online phase using rainbow tables, the computation trails are short in the beginning but increment in length over time. This results in significant congestion on COPACOBANA’s communication interface since a large number of table lookups are required in the beginning of the online phase. Therefore, a scenario running both the precomputation and the online phase on COPACOBANA should be based on the DP method since this method is most promising with respect to the restrictions of the machine.

We have implemented the precomputation phase for generating DES rainbow tables on COPACOBANA. For this implementation, we have developed another DES core that operates with 16 pipeline stages only.<sup>7</sup> Using four parallel DES units with 16 stages each, we can run 64 chain computations in parallel per FPGA. Fig. 6 graphically

6. Note that with tracking of start points in the host software, this must be indicated to the host PC to increment the  $SP$  counter accordingly.

7. Recall that the DES implementation in Section 3.1 uses 21 instead of 16 pipeline stages. A 16-stage implementation obviously allows for simpler addressing schemes when selecting a result from a specific pipeline position.

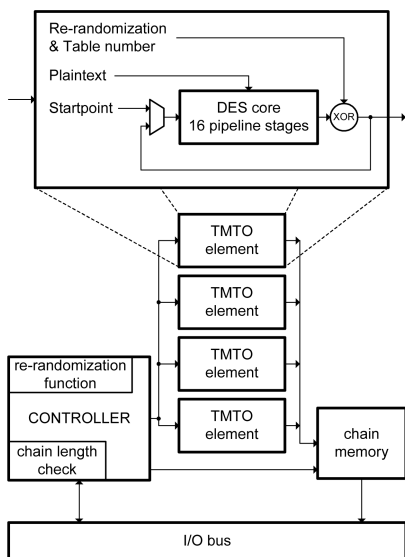


Fig. 6. Implementation for generating DES rainbow tables.

presents our architectures for generating rainbow tables in further detail. On the given Spartan-3 devices, our entire implementation, including I/O and control logic, consumes 7,571 out of 7,680 (98 percent) available slices of each FPGA and runs at a maximum clock frequency of 96 MHz. A single COPACOBANA is then able to compute more than 46 billion iterations of the step function  $f$  per second. We are currently optimizing the I/O logic to support concurrent trail computations *and* data transfers to eliminate idle times of the DES cores during data transmission. With this improvement of our design, we can estimate the actual duration of the precomputation phase for generating the rainbow tables to last slightly less than 32 days.

### 4.3 Time-Memory-Data Tradeoff Methods

The idea of cryptanalytic TMDTOs is due to [2] and [4]. TMDTOs are variants of TMTOs exploiting a scenario where multiple data points  $y_1, \dots, y_D$  of the function  $g$  are given and one has just to be successful in finding a preimage of one of them. Such a scenario typically arises in the cryptanalysis of stream ciphers, where we like to invert the function mapping the internal state (consisting of  $\log_2(N)$  bits) to the first  $\log_2(N)$  output bits of the cipher produced from this state. For an attack on a stream cipher, there are sometimes  $w > \log_2(N)$  bits of the output stream available. In this situation, it is possible to derive  $D = w - \log_2(N) + 1$  data points from the stream bits  $(b_1, \dots, b_w)$ , namely,  $y_1 = (b_1, b_2, \dots, b_{\log_2(N)})$ ,  $y_2 = (b_2, b_3, \dots, b_{\log_2(N)+1})$ , and so on. Thus, one has  $D$  chances to invert the function and “break” the cipher.

The common approach to exploit the existence of multiple data is to use an existing TMTO method and reduce the coverage of the tables by a factor of  $D$ , i.e., from the outset, one only aims to cover  $N/D$  points. Clearly, this has also effects on the precomputation and online complexity. The resulting scheme exhibiting the additional parameter  $D$  is then called a TMDTO. The adoption of Hellman’s method for a TMDTO on stream ciphers was first proposed and analyzed in [4]. Here, one can gain from reducing the number of tables.

**Thin-rainbow DP method.** As opposed to that, the plain rainbow scheme does not significantly gain from multiple data (by reducing the length of the chains), as recently shown in [3]. In the same paper, a new variant of the rainbow method, called *thin-rainbow method*, was sketched, providing a better TMDTO. In this variant, one does not use a different reduction function in each step of the chain computation but applies a sequence of  $S$  different reduction functions  $\ell$ -times periodically in order to generate a chain of length  $\ell S + 1$ . More precisely, the corresponding step functions  $f_1, \dots, f_S$  are applied in the following order:

$$f_1 f_2 \dots f_S f_1 f_2 \dots f_S \dots f_1 f_2 \dots f_S.$$

To reduce the number of disk accesses in the online phase, one can combine the thin-rainbow scheme with the DP method. This is done by looking for a DP after each application of the  $f_S$  function. During precomputation, we only save a chain if both this chain exhibits its first DP after  $\ell_{\min} \leq \ell \leq \ell_{\max}$  applications of  $f_S$ , for certain parameters  $\ell_{\min}$  and  $\ell_{\max}$ , and this DP is different from the end points of the chains already stored. (To get a better coverage, one usually stores the longer one of two chains with the same end point.) In the online phase, for each of the  $D$  given data points  $y_i$  one computes  $S$  chains. More precisely, from  $y_i$ , we derive the points  $R_1(y_i), \dots, R_S(y_i)$  and compute a chain for each of them until a DP is found (or  $\ell > \ell_{\max}$ ). An analysis of the characteristics of this method can be found in the Appendix.

### 4.4 A TMDTO Attack on A5/1

A5/1 is a synchronous stream cipher that is used for protecting GSM communication. In the GSM protocol, communication is organized in 114-bit frames that are encrypted by XORing them with 114-bit blocks of the keystream produced by the cipher as follows: A5/1 is based on three LFSRs, which are irregularly clocked. The three registers are 23, 22, and 19 bits long, representing the internal 64-bit state of the cipher. During initialization, a 64-bit key  $k$  is clocked in, followed by a 22-bit initialization vector that is derived from the publicly known frame number. After that, a warm-up phase is performed, where the cipher is clocked 100 times, and the output is discarded. Then, 228 bits of keystream are produced, which are partitioned into two blocks of 114 bits. One of them is used to encrypt the next frame carrying uplink traffic, while the other is used to decrypt the next incoming frame containing downlink traffic. For the following pair of uplink and downlink frames, the cipher is initialized with the same key and a new frame number. For a detailed description of A5/1, please refer to [5].

To make our attack realistic, we assume that a relatively small amount of only 114 consecutive bits of keystream is known. Hence, we have  $D = 114 - 64 + 1 = 51$  data points available to break the cipher.

#### 4.4.1 Chosen Method

Both Hellman’s original trade-off and rainbow tables are well suited for parallelization in hardware. Since the chains have a fixed length, the control of the calculation in the precomputation phase is simple. However, Hellman’s

method requires a large number of disk accesses during the online phase, while the rainbow table trade-off curve is inferior to Hellman's one whenever we have multiple data  $D$  [3]. The DP method significantly reduces the number of disk accesses, but the fact that the chain can reach its end point after *any* application of the step function hampers an efficient hardware implementation.

Hence, for our implementation, we have selected the thin-rainbow DP method described in the previous section. In the case of multiple data, this approach allows simple and efficient hardware implementation, while exhibiting a low number of disk accesses during the online phase and an efficient trade-off curve.

#### 4.4.2 Design Approach

Most designs realized on FPGAs usually do not fully use the flip-flops available on the chip. Typically, the designs are limited by the number of combinational resources (LUTs) available. In the case of A5/1, it is different: a demand for flip-flops prevails over a demand for combinational logic. Fortunately, some LUTs in the Spartan-3 chips [48] can be configured to work as a shift register with a maximum length of 16 bits (denoted as SRL16). This property enables us to implement much bigger shift-register-based circuits<sup>8</sup> under the condition that some limitations to circuit design are satisfied. To allow the synthesis tool to utilize this property, we have to avoid any parallel input to the register, using only serial inputs and outputs. Hence, we decided to implement an array of small independent processing units (we call them *TMTO elements*) with serial inputs and outputs rather than to create the pipeline like in the case of DES engines. To gain the maximum frequency, we rejected the idea of parallel access to the TMTO elements, since the number of them is relatively large. Instead of that, we connected all TMTO elements into one large chain (see Fig. 7).

#### 4.4.3 How It Works

Each TMTO element is calculating one chain of points, i.e., one row in the TMTO table. Each element consists of two coupled A5/1 cores. In odd steps of rainbow sequences, Core 1 produces a 64-bit block  $s$  of keystream that is rerandomized and loaded to Core 2 as the new internal state. In even steps, the functionalities of the cores are swapped. As a source of rerandomization, we use the long-period LFSR, whose output is XORed with  $s$ .

First, all TMTO elements are initialized with start points. Then, the rainbow sequences are performed. After each rainbow sequence  $f_1, f_2, \dots, f_S$ , the result in each element is checked for the DP criterion. If a DP has been reached, the chain information is stored in FIFO, and the computation of the new chain is started in the element. If no DP has been reached yet, another rainbow sequence of the chain is performed. If the chain becomes too long or if it is too short, the result is discarded, and the calculation of the new chain is started too. Information from FIFO is

8. For example, Xilinx Spartan 3-1000 contains 15,360 flip-flops that can be used to implement less than  $15,360/64 = 240$  A5/1 cores, since some flip-flops will also be used for other necessary units. Using SRL16s, we can implement up to 480 A5/1 cores, still leaving enough LUTs and flip-flops for controller and other circuits.

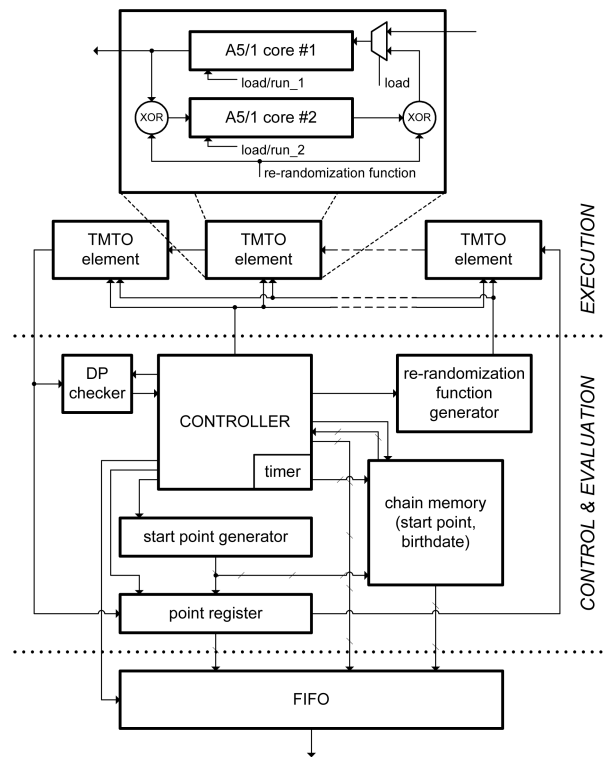


Fig. 7. Overview of an FPGA with an A5/1 TMTO engine.

periodically read by the host computer and is stored on the disk.

#### 4.4.4 Implementation Results

An A5/1 TMTO engine can currently run at a maximum frequency of 156 MHz. Computing a step function  $f_i$  takes 64 clock cycles. One FPGA contains 234 TMTO elements (each consisting of two A5/1 cores); hence, the whole COPACOBANA can perform approximately  $2^{36}$  step functions per second.

To select the TMTO parameters (like the length  $S$  of the rainbow sequence, the number  $d$  of bits defining the DP criterion, the interval  $I_t = [l_{min}, l_{max}]$  defining the minimum and maximum number of rainbow sequence applications, and the number  $m$  of start points) requires special attention, since this highly influences the precomputation time (PT), the disk usage (DU), the time needed in the online phase for the chain computations (OT), the number of table accesses (TA), and the success rate (SR). Table 4 summarizes the results for different sets of parameter choices. The estimations are based on the analysis presented in the Appendix under the assumption that  $D = 64$ . Furthermore, we assumed that COPACOBANA is used not only for the precomputation but also for the online phase. Due to this, it is worth trading higher online complexity, e.g., for lower demand for disk space (compare rows 4 and 5). For our implementation, we have selected the set of parameters presented in the third row, since it produces a reasonable precomputation time and a reasonable size of the tables, as well as a relatively small number of table accesses. The success rate of 63 percent may seem to be small, but it increases significantly if more data samples are available. For instance, if four frames of known keystream are

TABLE 4  
A5/1 TMDTO: Expected Runtimes and Memory Requirements

$m$	$S$	$d$	$I_\ell$	PT [days]	DU [TB]	OT [secs]	TA	SR
$2^{41}$	$2^{15}$	5	$[2^3, 2^6]$	337.5	7.49	27.8	$2^{21}$	0.86
$2^{39}$	$2^{15}$	5	$[2^3, 2^7]$	95.4	3.25	36.3	$2^{21}$	0.67
$2^{40}$	$2^{14}$	5	$[2^4, 2^7]$	95.4	4.85	10.9	$2^{20}$	0.63
$2^{40}$	$2^{14}$	5	$[2^3, 2^6]$	84.4	7.04	7.0	$2^{20}$	0.60
$2^{39}$	$2^{15}$	5	$[2^3, 2^6]$	84.4	3.48	27.8	$2^{21}$	0.60
$2^{40}$	$2^{14}$	5	$[2^4, 2^6]$	84.4	5.06	8.5	$2^{20}$	0.55
$2^{37}$	$2^{15}$	6	$[2^4, 2^8]$	47.7	0.79	73.5	$2^{21}$	0.42

available, then  $D = 4 \cdot 51 = 204$ , and thus, the success rate is increased to 96 percent.

## 5 INTEGER FACTORIZATION

The factorization of a large composite integer  $n$  is a well-known mathematical problem that has attracted special attention since the invention of public-key cryptography. RSA is known as the most popular asymmetric cryptosystem and was originally developed by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977 [41]. Since the security of RSA relies on the attacker's inability to factor large numbers, the development of a fast factorization method could allow for cryptanalysis of RSA messages and signatures. Recently, the best known method for factoring large integers is the General Number-Field Sieve (GNFS). An important step in the GNFS algorithm is the factorization of mid-sized numbers for smoothness testing. For this purpose, the ECM has been proposed by Lenstra [30], which has been proved to be suitable for parallel hardware architectures in [9], [15], and [44], particularly on FPGAs.

The ECM algorithm performs a very high number of operations on a very small set of input data and is not demanding in terms of high communication bandwidth. Furthermore, it requires only little memory. The operands required 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 solution.

In [9], it has been shown that the utilization of DSP slices in Virtex-4 and -5 FPGAs for implementing a Montgomery multiplication can significantly improve the ECM performance. In this contribution, the authors used a fully parallel multiplier implementation that provides the best known performance figures so far but still does not exploit the full potential of the Virtex-4 FPGAs.

Based on this approach, we designed a new slot-in module for use with a second release of COPACOBANA, hosting eight Xilinx Virtex-4 XC4V5X35 FPGAs, each providing 192 DSP slices. Due to the larger size of the FPGAs (FF668 package with dimension of  $27 \times 27$  mm), we enlarged the modules, which includes also modifications of the corresponding connectors on the backplane. For more efficient heat dissipation at high clock frequencies up to 400 MHz, an actively ventilated heat sink is attached to each FPGA. With a more powerful power supply providing 1.5 kW at 12 V, we are able to run a total of 128 Virtex-4 SX35 FPGAs distributed over 16 plug-in modules. In contrast to [9], we used a multicore ECM design per FPGA.

TABLE 5  
Clock Cycles and Frequency for Point Multiplication of 151-Bit Numbers Required in Phase 1 of ECM

Aspect	This work	[15]
Modular Adder/Subtractor	24 clk	31 clk <sup>9</sup>
Montgomery Multiplication	66 clk	167 clk
Point Doubling	287 clk	n/a
Parallelized Point Doubling and Addition	377 clk	947 clk
Clock Frequency of an ECM Core	171 MHz	135 MHz

<sup>9</sup> The presented cycle count for 151-bit modular addition was estimated based on the results given in [15] for 198-bit parameters.

A single ECM engine comprises of an arithmetic unit computing modular multiplication and additions, a point multiplication unit for phase 1, and ROM tables for phase 2. At this point of development, we can provide figures, shown in Table 5, for the most relevant units and compare our results to the implementation presented in [15].

## 6 COMPUTING ELLIPTIC CURVE DISCRETE LOGARITHMS

Another popular problem used for building public-key cryptosystems is known as the Discrete Logarithm Problem (DLP), where the exponent  $\ell$  should be determined for a given  $a^\ell \bmod n$ . A popular derivative is the ECDLP for ECCs [18].

An attack on ECC relies on the same algorithmic primitives as the cryptosystem itself, namely, point addition and point doubling. Up to now, the best known algorithm for this purpose is the PR algorithm for parallel implementation described in [47]. This variant of the original PR method [40] allows for a linear gain in performance with the number of available processors. This can be efficiently implemented in hardware, as presented in [16].

The PR algorithm essentially determines DPs on the elliptic curve. These points are reported to a central host computer, which awaits a collision of two points. Like with TMDTOs (cf. Section 4.1), a DP is defined to be a point with a specific characteristic, e.g., its  $x$ -coordinate has a fixed number of leading zero bits. To reach such a DP, PR follows a so-called pseudorandom walk on the elliptic curve by subsequently adding points from a finite set of random previously defined points. Hence, with careful parameterization of the DP criterion, the duration of a computation until a DP is found can be adapted to the bandwidth constraints of the system. Furthermore, the PR does not need a large memory for computation so that the COPACOBANA system seems to be a suitable platform for running the algorithm. As with the ECM unit, a single PR unit is comprised of an arithmetic unit, a few kilobytes of RAM, and control logic. The arithmetic unit supports modular inversion as an additional function required for uniquely determining DPs.

For a parallelized PR on COPACOBANA according to the method presented in [47], all instances of the algorithm can run completely independent of each other. For solving the DLP over curves defined over prime fields  $\mathbb{F}_p$ , we have to compute approximately  $\sqrt{q}$  points, where  $q$  is the largest prime power of the order of the curve. Note that the transfer of data between the host computer and point processing

TABLE 6  
Expected Runtime on Different Platforms  
and for Different Certicom ECC Challenges

$k$	Certicom Est. [7]	XC3S1000	COPACOBANA
79	146 d	15.3 d	3.06 h
89	12.0 y	1.62 y	4.93 d
97	197 y	30.7 y	93.4 d
109	$2.47 \cdot 10^4$ y	$2.91 \cdot 10^3$ y	24.3 y
131	$6.30 \cdot 10^7$ y	$7.40 \cdot 10^6$ y	$6.17 \cdot 10^4$ y
163	$6.30 \cdot 10^{12}$ y	$9.15 \cdot 10^{11}$ y	$7.62 \cdot 10^9$ y

units on the FPGA can be performed independently from the computations.

Implementing the PR on Spartan-3 FPGAs for solving the ECDLP over curves with a length of 160 bits, we achieve a maximum clock frequency of approximately 40 MHz and an area usage of 6,067 slices (79 percent) for two parallel instances. The corresponding point addition requires 846 cycles so that slightly less than 50,000 point operations can be performed per second by one unit. Consequently, a single COPACOBANA can compute about 11.3 million points per second. Table 6 compares our results for COPACOBANA with challenges and corresponding estimates from Certicom based on the computing time of an Intel Pentium 100. Obviously, elliptic curves as proposed by SECG [1] with bit lengths of less than 100 bits do not offer more protection than a few days against an attack using a single COPACOBANA.

## 7 CONCLUSION

In this work, we presented novel implementations for cryptanalytical applications on COPACOBANA. On up to 120 low-cost FPGAs, COPACOBANA is able to perform cryptographic operations simultaneously and in parallel for applications with high computational but low memory and communication requirements.

We demonstrated how the DES can be broken within less than a week at an average throughput of 65.3 billion searched keys per second. Besides a simple brute-force scenario on DES, we have extended the attack scheme for tackling the complexity of ANSI X9.9 OTP tokens and Norton Diskreet whose security assumptions rely on the DES.

Furthermore, we presented a successful attack on the BAC scheme used for securing private data and establishing a confidential wireless channel for the communication with international e-passports. Our attack is able to reveal BAC keys for encryption and authentication in real time for practical scenarios, due to the low entropy of the keys.

Smarter brute-force attacks, particularly when we are frequently faced with the encryption of a fixed plaintext under different keys, can be achieved by TMTO and TMDTO. We suggested two options to utilize COPACOBANA for TMTO and TMDTO attacks on the DES and the A5/1.

Besides the symmetric cryptography, we can use COPACOBANA to attack public-key cryptosystems. We proposed a massively parallel implementation of the ECM for factoring mid-sized integers typically obtained from the GNFS for RSA factorization. Finally, we analyzed the security of ECCs by solving the ECDLP with a COPACOBANA-based architecture of the parallel PR algorithm.

## APPENDIX

### CHARACTERISTICS OF THE THIN-RAINBOW DP METHOD

In the following, we analyze the thin-rainbow DP method described in Section 4.3.

**Success probability.** Let us first assume that we compute  $m$  thin-rainbow chains, each of *fixed* length  $\ell S + 1$ , i.e., without applying the DP method and without rejecting merging chains. The resulting thin-rainbow table has the following structure:

$$\begin{array}{ccccccc} x_{1,1} & \xrightarrow{f_1} & \dots & \xrightarrow{f_s} & x_{1,S+1} & \dots & x_{1,\ell S-S+1} & \xrightarrow{f_1} & \dots & \xrightarrow{f_s} & x_{1,\ell S+1}, \\ & & & & & & & & & & \vdots \\ x_{m,1} & \xrightarrow{f_1} & \dots & \xrightarrow{f_s} & x_{m,S+1} & \dots & x_{m,\ell S-S+1} & \xrightarrow{f_1} & \dots & \xrightarrow{f_s} & x_{m,\ell S+1}. \end{array}$$

In the following, we estimate the coverage of such a table (in a similar way as done in [36] for the rainbow scheme). Thereby, we ignore the slight reduction of the coverage due to colliding but nonmerging chains. Let  $m_i$  denote the expected number of new distinct points in column  $i$  of the thin-rainbow table, where “new” means that these points did not occur in the previous columns  $i - S, i - 2S, \dots, i - \lfloor \frac{i}{S} \rfloor S$ . Note that  $m_i$  also corresponds to the expected number of chains that did not merge until and including column  $i$ . Clearly, we have  $m_1 = m$ , and we set  $m_0 = 0$ . To determine  $m_i$  for  $1 < i \leq \ell S + 1$  (recursively), we make use of the indicator variables  $X_j^{(i)}$  for  $0 \leq j \leq N - 1$ , where

$$X_j^{(i)} = \begin{cases} 1, & \text{point } j \text{ occurs not in cols } c \leq i, c \equiv i \pmod{S}, \\ 0, & \text{else.} \end{cases}$$

Then, we have  $\Pr[X_j^{(i)} = 1] = (1 - \frac{1}{N} \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS})(1 - \frac{1}{N})^{m_{i-1}}$ , and we can calculate the number of new distinct points in column  $i$  as

$$\begin{aligned} m_i &= N - E\left(\sum_{j=0}^{N-1} X_j^{(i)}\right) - \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS} \\ &= N - \sum_{j=0}^{N-1} \Pr[X_j^{(i)} = 1] - \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS} \\ &= \left(N - \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS}\right) \left(1 - \left(1 - \frac{1}{N}\right)^{m_{i-1}}\right) \\ &\approx \left(N - \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS}\right) \left(1 - e^{-\frac{m_{i-1}}{N}}\right), \end{aligned} \quad (2)$$

where  $E(\cdot)$  denotes the expectation. Hence, the probability that a random point is contained in a table generated by the plain thin-rainbow scheme can be approximated by  $1 - \prod_{i=1}^{\ell S+1} (1 - \frac{m_i}{N})$ , where  $m_0 = 0$ ,  $m_1 = m$ , and  $m_i = (N - \sum_{k=1}^{\lfloor \frac{i}{S} \rfloor} m_{i-kS})(1 - e^{-\frac{m_{i-1}}{N}})$  for  $i > 1$ .

Next, let us consider the combination of the thin-rainbow and DP scheme as described in Section 4.1. Here, we look for a DP after each application of the  $f_S$  function. During precomputation we only save a chain if it is of length  $\ell_{\min} S + 1 \leq t \leq \ell_{\max} S + 1$  and it ends in a DP not occurred before. To estimate the success probability of this modified

scheme we first determine the number  $m' \leq m$  of chains exhibiting their (first) DP after  $\ell \in [\ell_{\min}, \ell_{\max}]$  applications of the  $f_S$  function. Note that only this fraction is processed further and all other chains are immediately discarded during precomputation. Then we calculate the average number of  $f_S$  applications, denoted by  $\ell_{av}$ , required for the remaining chains.

To determine  $m'$  and  $\ell_{av}$  we follow the approach in [45]. Let the DP criterion be a bit mask of length  $d$  and let  $N = 2^k$ . Furthermore, by  $P_{DP}(\ell)$  we denote the probability that a DP is reached after at most  $\ell$  applications of the  $f_S$  function. Clearly, we have  $P_{DP}(\ell) \approx 1 - (1 - \frac{1}{2^d})^\ell$ .

The probability to find a DP after at least  $\ell_{\min}$  and at most  $\ell_{\max}$  iterations is given by

$$\begin{aligned} & \Pr[\text{DP in } \ell_{\min} \leq \ell \leq \ell_{\max} \text{ iterations}] \\ &= P_{DP}(\ell_{\max}) - P_{DP}(\ell_{\min} - 1) \\ &\approx \left(1 - \frac{1}{2^d}\right)^{\ell_{\min}-1} - \left(1 - \frac{1}{2^d}\right)^{\ell_{\max}}. \end{aligned}$$

This immediately yields an approximation for the expected number of chains with a length in the desired range:

$$\begin{aligned} m' &= \sum_{i=1}^m \Pr[\text{DP in } \ell_{\min} \leq \ell \leq \ell_{\max} \text{ iterations}] \\ &\approx m \left( \left(1 - \frac{1}{2^d}\right)^{\ell_{\min}-1} - \left(1 - \frac{1}{2^d}\right)^{\ell_{\max}} \right). \end{aligned}$$

Similarly, the average number of  $f_S$  applications required for these chains can be approximated:

$$\begin{aligned} \ell_{av} &= \frac{1}{m'} \sum_{\ell=\ell_{\min}}^{\ell_{\max}} m(P_{DP}(\ell) - P_{DP}(\ell-1))\ell \\ &\approx \frac{\sum_{\ell=\ell_{\min}}^{\ell_{\max}} \left( \left(1 - \frac{1}{2^d}\right)^{\ell-1} - \left(1 - \frac{1}{2^d}\right)^\ell \right) \ell}{\left(1 - \frac{1}{2^d}\right)^{\ell_{\min}-1} - \left(1 - \frac{1}{2^d}\right)^{\ell_{\max}}}. \end{aligned}$$

Now, we assume that we have a thin-rainbow table of dimension  $m' \times (\ell_{av}S + 1)$ . From (2), we know the number  $\hat{m}$  of nonmerging chains of this table since this number is equal to  $\hat{m} = m\ell_{av}S + 1$ , where we start with  $m_1 = m'$  points. Note that not  $m'$  but  $\hat{m}$  is the number of chains in our final table since merging chains are also sorted out in the precomputation phase. Finally, we estimate the success probability as  $P_{succ} \approx \frac{\hat{m}\ell_{av}S}{N}$ , where we neglect the fact that the average chain length slightly decreases by sorting out merging chains [45] (since the fraction of merging chains that are longer than the average is slightly higher). Furthermore, as it is usually done, our estimation does not take into account that a point occurring in column  $i$  of the table could also occur (undetected) in columns  $c \not\equiv i \pmod{S}$ . Since we are successful if at least one of the  $D$  given points is covered by the table, we get a total success probability of

$$P_{succ} \approx 1 - \left(1 - \frac{\hat{m}\ell_{av}S}{N}\right)^D.$$

**Disk usage.** After sorting out merging chains and chains that do not comply to the length restrictions, we need to

store  $\hat{m}$  triples. Each triple consists of the start point, the nondistinguished part of the end point, and the required number  $0 \leq \ell \leq \ell_{\max} - \ell_{\min}$  of  $f_S$  applications after reaching  $\ell_{\min}$ . Thus, we have to store

$$M \approx \hat{m}(\lceil \log_2(m) \rceil + k - d + \lceil \log_2(\ell_{\max} - \ell_{\min} + 1) \rceil)$$

bits on a hard disk.

**Precomputation time.** We continue a chain until either a DP is reached or its length equals  $S\ell_{\max} + 1$ . Thus, the expected number of iterations is

$$\begin{aligned} z &= \ell_{\max}(1 - P_{DP}(\ell_{\max})) + \sum_{\ell=1}^{\ell_{\max}} \ell(P_{DP}(\ell) - P_{DP}(\ell-1)) \\ &\leq \ell_{\max}(1 - P_{DP}(\ell_{\max})) + \ell_{av}P_{DP}(\ell_{\max}). \end{aligned}$$

Since we compute  $m$  chains

$$T_{pr} \approx mSz$$

steps are expected for the precomputation phase.

**Online time.** In the online phase, we compute at most  $S$  chains for each of the  $D$  given points. To compute such a chain and verify if the wanted point is included in the table, we need about  $\ell_{av}S$  steps. Thus, the total number of steps can be estimated as

$$T_{on} \approx DS\ell_{av}S.$$

Finally, since for each computed chain, we need to access the table at most once, the total number of disk accesses is bounded by

$$A = DS.$$

## ACKNOWLEDGMENTS

The authors would like to thank Jean-Jacques Quisquater, François-Xavier Standaert (UCL), Gerd Pfeiffer and Manfred Schimmler (University Kiel), as well as Jan Pelzl, Kerstin Lemke-Rust and Stefan Spitz, for their tremendous help on our work with COPACOBANA and its applications.

## REFERENCES

- [1] *Standards for Efficient Cryptography—SEC 1: Elliptic Curve Cryptography*, [http://www.secg.org/secg\\_docs.htm](http://www.secg.org/secg_docs.htm), Sept. 2000.
- [2] S. Babbage, "A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers," *Proc. European Convention Security and Detection*, vol. 408, 1995.
- [3] E. Barkan, E. Biham, and A. Shamir, "Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs," *Proc. 26th Ann. Int'l Cryptology Conf. (CRYPTO '06)*, pp. 1-21, 2006.
- [4] A. Biryukov and A. Shamir, "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers," *Proc. Sixth Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT '00)*, pp. 1-13, 2000.
- [5] A. Biryukov, A. Shamir, and D. Wagner, "Real Time Cryptanalysis of A5/1 on a PC," *Proc. Eighth Int'l Workshop Fast Software Encryption (FSE '00)*, pp. 1-18, 2001.
- [6] D. Carluccio, K. Lemke-Rust, C. Paar, and A.-R. Sadeghi, "E-Passport: The Global Traceability or How to Feel Like an UPS Package," *Proc. Seventh Int'l Workshop Information Security Applications (WISA '06)*, pp. 391-404, 2006.
- [7] Certicom Corp., Certicom ECC Challenges, <http://www.certicom.com>, 2005.
- [8] D. Denning, *Cryptography and Data Security*. Addison-Wesley, 1982.

- [9] G. de Meulenaer, F. Gosset, M.M. de Dormale, and J.-J. Quisqater, "Integer Factorization Based on Elliptic Curve Method: Towards Better Exploitation of Reconfigurable Hardware," *Proc. 15th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM '07)*, pp. 197-206, 2007.
- [10] W. Diffie and M.E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, vol. 10, no. 6, pp. 74-84, June 1977.
- [11] Electronic Frontier Foundation, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, July 1998.
- [12] Germany Fed. Office for Information Security, *Advanced Security Mechanisms for Machine Readable Travel Documents—Extended Access Control*, [http://www.bsi.de/fachthem/epass/EACTR03110\\_v110.pdf](http://www.bsi.de/fachthem/epass/EACTR03110_v110.pdf), 2007.
- [13] T. Finke and H. Kelter, "Radio Frequency Identification—Abhörmöglichkeiten der Kommunikation zwischen Lesegerät und Transponder am Beispiel eines ISO14443-Systems," [http://www.bsi.de/fachthem/rfid/Abh\\_RFID.pdf](http://www.bsi.de/fachthem/rfid/Abh_RFID.pdf), 2007.
- [14] K. Finkenzerler, *RFID-Handbook*. John Wiley & Sons, 2003.
- [15] K. Gaj, S. Kwon, P. Baier, P. Kohlbrenner, H. Le, M. Khaleeluddin, and R. Bachimanchi, "Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware," *Proc. Eighth Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '06)*, pp. 119-133, 2006.
- [16] T. Gueneysu, C. Paar, and J. Pelzl, "Attacking Elliptic Curve Cryptosystems with Special-Purpose Hardware," *Proc. 15th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA '07)*, pp. 207-215, 2007.
- [17] G.P. Hancke, "Practical Attacks on Proximity Identification Systems (Short Paper)," *Proc. IEEE Symp. Security and Privacy (SP '06)*, pp. 328-333, 2006.
- [18] D.R. Hankerson, A.J. Menezes, and S.A. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.
- [19] M.E. Hellman, "A Cryptanalytic Time-Memory Trade-Off," *IEEE Trans. Information Theory*, vol. 26, pp. 401-406, 1980.
- [20] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. Wichers Schreur, "Crossing Borders: Security and Privacy Issues of the European E-passport," *Proc. First Int'l Workshop Security (IWSEC '06)*, pp. 152-167, 2006.
- [21] ICAO, "Machine Readable Travel Documents, PKI for Machine Readable Travel Documents Offering ICC Read-Only Access," technical report, <http://www.mrtd.icao.int>, 2004.
- [22] ICAO, Machine Readable Travel Documents, Supplement to Doc9303-Part1-Sixth Edition, 2005.
- [23] ICAO, Machine Readable Travel Documents, Doc 9303, Part 1 Machine Readable Passports, fifth ed., 2003.
- [24] ISO/IEC 14443, Identification Cards—Contactless Integrated Circuit(s) Cards—Proximity Cards—Part 1-4, [www.iso.ch](http://www.iso.ch), 2001.
- [25] S. Vaudenay, J. Monnerat, and M. Vuagnoux, "About Machine-Readable Travel Documents," *Proc. Third Conf. RFID Security (RFIDSec '07)*, pp. 15-28, 2007.
- [26] A. Juels, D. Molnar, and D. Wagner, "Security and Privacy Issues in E-passports," *Proc. First Int'l Conf. Security and Privacy for Emerging Areas in Comm. Networks (SecureComm '05)*, pp. 74-88, 2005.
- [27] T. Kasper, D. Carluccio, and C. Paar, "An Embedded System for Practical Security Analysis of Contactless Smartcards," *Proc. Workshop Information Theory and Practice (WISTP '07)*, pp. 150-160, 2007.
- [28] G.S. Kc and P.A. Karger, "Security and Privacy Issues in Machine Readable Travel Documents (MRTDs)," RC 23575, IBM T.J. Watson Research Labs, Apr. 2005.
- [29] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking Ciphers with COPACOBANA—A Cost-Optimized Parallel Code Breaker," *Proc. Eighth Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '06)*, pp. 101-118, 2006.
- [30] H. Lenstra, "Factoring Integers with Elliptic Curves," *Annals of Math.*, vol. 126, pp. 649-673, 1987.
- [31] Y. Liu, T. Kasper, K. Lemke-Rust, and C. Paar, "E-Passport: Cracking Basic Access Control Keys," *Proc. On the Move to Meaningful Internet Systems Workshops (OTM '07) Part II*, pp. 1531-1547, 2007.
- [32] Int'l Business Machines, IBM Research: BlueGene, <http://www.research.ibm.com/bluegene/>, 2007.
- [33] N. Mentens, L. Batina, B. Prenel, and I. Verbauwhede, "Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking," *Proc. Int'l Workshop Applied Reconfigurable Computing (ARC '06)*, pp. 323-334, 2006.
- [34] ICAO TAG MRTD/NTWG, "Biometrics Deployment of Machine Readable Travel Documents," technical report, 2004.
- [35] NIST FIPS PUB 46-3, *Data Encryption Standard*, Fed. Information Processing Standards, Nat'l Bureau of Standards, US Dept. of Commerce, Jan. 1977.
- [36] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," *Proc. 23rd Ann. Int'l Cryptology Conf. (CRYPTO '03)*, pp. 617-630, 2003.
- [37] Nat'l Inst. of Standards and Technology, *FIPS 180-3 Secure Hash Standard (Draft)*, <http://www.csrc.nist.gov/publications/PubsFIPS.html>, 2007.
- [38] P. Gutmann, *Norton's InDiskreet*, posting to sci.crypt newsgroup, Nov. 1993.
- [39] P. Kocher, *Norton Diskreet (Security Overview)*, posting to sci.crypt newsgroup, Nov. 1993.
- [40] J.M. Pollard, "Monte Carlo Methods for Index Computation mod  $p$ ," *Math. Computation*, vol. 32, no. 143, pp. 918-924, July 1978.
- [41] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, vol. 21, no. 2, pp. 120-126, Feb. 1978.
- [42] H. Robroch, "ePassport Privacy Attack," presentation at Cards Asia Singapore, <http://www.riscure.com>, Apr. 2006.
- [43] G. Rouvroy, F.-X. Standaert, J.-J. Quisqater, and J.-D. Legat, "Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES," *Proc. 11th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA '03)*, p. 247, 2003.
- [44] M. Šimka, J. Pelzl, T. Kleinjung, J. Franke, C. Priplata, C. Stahlke, M. Drutarovský, V. Fischer, and C. Paar, "Hardware Factorization Based on Elliptic Curve Method," *Proc. 13th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 107-116, 2005.
- [45] F. Standaert, G. Rouvroy, J. Quisqater, and J. Legat, "A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results," *Proc. Fourth Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '02)*, pp. 596-611, 2002.
- [46] Univ. of California, Berkeley, *Seti@Home Website*, <http://setiathome.berkeley.edu/>, 2005.
- [47] P.C. van Oorschot and M.J. Wiener, "Parallel Collision Search with Cryptanalytic Applications," *J. Cryptology*, vol. 12, no. 1, pp. 1-28, 1999.
- [48] Xilinx, *Spartan-3 FPGA Family: Complete Data Sheet, DS099*, <http://www.xilinx.com>, Jan. 2005.



**Tim Güneysu** has studied international information technology and IT security at the University of Cooperative Education Mannheim, Ruhr-University Bochum, Germany, and the University of Stafford, United Kingdom. He received degrees in 2003 and 2006. He is currently a research assistant for the chair for communication security at the Horst Görtz Institute for IT-Security, Ruhr-University Bochum. His field of research is mainly focused on the implementa-

tion of asymmetric cryptographic implementations and cryptanalysis with special-purpose hardware.



**Timo Kasper** has studied electrical engineering and information security at Ruhr-University Bochum, Germany, and the University of Sheffield, United Kingdom. He became a graduate engineer in 2006. He is currently a research assistant in the Communication Security Group, Horst Görtz Institute for IT-Security, Ruhr-University Bochum. His field of research covers the security of smart cards, RFID, and wireless communication, as well as side-channel cryptanalysis and the security of embedded systems.



**Martin Novotný** received the master's degree in computer science and engineering from the Czech Technical University, Prague, in 1992. Currently, he is a PhD student at the Czech Technical University and at the Horst Görtz Institute for IT-Security, Ruhr-University Bochum. His research interests include embedded systems, digital design, arithmetic units, cryptanalytical hardware and efficient hardware implementation of cryptographic algorithms.



**Christof Paar** received the PhD degree in electrical engineering from the Institute for Experimental Mathematics, University of Essen. He holds the chair of communication security in the Electrical and Computer Engineering Department, Horst Görtz Institute for IT-Security, Ruhr-University Bochum. His research interests include physical security, cryptanalytical hardware, security in real-world systems, and efficient software and hardware implementations of cryptographic algorithms. He is a member of the IEEE, the ACM, and the International Association for Cryptologic Research (IACR).



**Andy Rupp** received the master's degree in computer science from Saarland University, Saarbrücken, Germany, in 2004. Currently, he is a PhD student at the Horst Görtz Institute for IT-Security, Ruhr-University Bochum, under the supervision of Christof Paar, the chair of communication security. His research interests include theoretical aspects of cryptography like cryptographic assumptions and models of computation as well as practical aspects like special-purpose hardware for cryptography and cryptanalysis.

▷ **For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).**