How to Break DES for € 8,980

Sandeep Kumar[1], Christof Paar[1], Jan Pelzl[1],
Gerd Pfeiffer[2], Andy Rupp[1], Manfred Schimmler[2]

[1] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{kumar,cpaar,pelzl,arupp}@crypto.rub.de

[2] Institute of Computer Science and Applied Mathematics, Faculty of Engineering,
Christian-Albrechts-University of Kiel, Germany
{gp,masch}@informatik.uni-kiel.de

Abstract

Cryptanalysis of symmetric and asymmetric ciphers is computationally extremely demanding. Since the security parameters of almost all practical crypto algorithms are chosen such that attacks with conventional computers are computationally infeasible, the only promising way to tackle existing ciphers (in the absence of mathematical breakthroughs) is to build special-purpose hardware. Dedicating those machines to the task of cryptanalysis holds the promise of a dramatically improved cost-performance ratio so that breaking of commercial ciphers comes within reach.

This contribution describes the design and realization of the reprogrammable machine COPACOBANA (Cost-Optimized Parallel Code Breaker), which is optimized for running cryptanalytical algorithms. The primary design goal was to produce a re-programmable low-cost design for less than € 10,000 which is applicable for attacking the Data Encryption Standard (DES) in less than nine days.

It will be shown that the architecture outperforms conventional computers by several orders of magnitude. Fully configured, COPACOBANA hosts 120 low-cost FPGAs and is able to perform an exhaustive key search of DES at a rate of more than 2^{35} keys per second, yielding an average search time of less than nine days. For this, we used the high-speed DES engine design of the Université Catholique de Louvain's Crypto Group.

We provide a real-world example by giving an estimate of an attack with COPACOBANA against a formerly popular encryption tool (Norton Diskreet). Due to a cryptographical weak key derivation function it can be broken in very little time by applying a smart key search. As a further application, COPACOBANA can also be used to attack machine readable travel documents (ePass).

COPACOBANA is suitable for computational problems which are parallelizable and have low communication requirements. The hardware can be used, e.g., to attack elliptic curve cryptosystems and to factor numbers. COPACOBANA is intended to, but not necessarily restricted to solving problems related to cryptanalysis.

1 Introduction

All modern practical ciphers, both symmetric and asymmetric ones, use security parameters (in particular the key length) providing an adjustable security margin against computational attacks with current computers. Depending on the security margin chosen in a given application, many ciphers are potentially vulnerable to attacks with special-purpose machines which have, say, a cost-performance ratio which is by a factor of 2^{10} to 2^{20} (three to six orders of magnitude) better than that of current PCs. This contribution describes a design and successful prototype realization of such a special-purpose cryptanalytical machine based on low-cost FPGAs.

Reconfigurable computing has been emerged as a cost effective alternative for various applications which require the power of a custom hardware but require the flexibility provided by a software based design, e.g., in rapid prototyping. Cryptanalysis of modern cryptographic algorithms requires massive computational effort, often between 2^{56} to 2^{80} operations. A characteristic of many (but not all) cryptanalytical algorithms is that they can run in a highly parallel fashion with very little interprocess communication. Such applications map naturally to a hardware based design, requiring repetitive mapping of the basic block, and can be easily extended by adding more chips as required. However, it should be stressed that the mere availability of computational resources is not the core problem, but providing massive computational resources *at affordable costs* is. The non recurring engineering costs for ASICs have put special-purpose hardware for cryptanalysis in almost all practical situations out of reach for commercial or research institutions, and have been considered only feasible by government agencies.

An alternative approach to distributed computing with loosely coupled processors is based on using the idle cycles of the huge number of computers connected via the Internet, for instance the SETI@home project [21]. The results of this approach has been quite successful for some applications (even though the confirmed detection of extraterrestrial life is still an open problem) and is used for selected problems which are not viable with the computing power within a single organization. Using distributed computing, however, has the disadvantage of, first, having to find individuals who would be interested in joining to solve a problem and, secondly, trusting the nodes from introducing errors. Finally, for many code-breaking application, shared computation is not a method of choice in many cases.

In cryptanalysis, certain algorithms are very well suited for special-purpose hardware. A prime example for this is an exhaustive key search of the Data Encryption Standard (DES) [11]. Such a brute-force attack is more than two orders of magnitude faster when implemented on FPGAs than in software on general purpose computers at equivalent costs[1]. However, for most crypto algorithms the cost-performance advantage of special-purpose hardware over general-purpose CPUs is not quite as dramatic as in the case of DES, particular for public-key algorithms.

With the recent advent of low-cost FPGA families with much logic resources, field programmable gate arrays provide a very interesting alternative tool for the massive computational effort required for cryptanalytic applications. In addition to the cost-performance advantage over PC-based machines, such a machine has the advantage over ASIC-based designs that it can be used to attack various different cryptosystems without the need to rebuilt a new machine each time. This contribution describes the design, implementation, and applications of COPACOBANA[2], a massively parallel machine based on FPGAs.

In the next Section, we present a model for an optimized hardware architecture for breaking codes which we realized as a custom-designed computing machine. We will present the architectural concept and the realization of COPACOBANA, consisting of a backplane, 20 FPGA DIMM modules, and a controller card. In Section 3, we will discuss the architecture for an exhaustive key search of DES, which is perfectly suited for running on low-cost FPGAs. The

[1]Based on our implementational results, a single FPGA at a cost of € 40 (current market price) can test 400 million keys per second, a PC (Pentium4, 3GHz) for € 200 checks 2 million keys per second. Hence, 5 FPGAs can perform the same task approximately 1000 times faster than a PC at the same cost.

[2]Yes, we know, Rio de Janeiro's famous beach is spelled slightly differently, Copacabana, but coming up with the current name has made us already painfully aware of our limited abilities in the imaginative area.

implementation is capable of searching the complete key space at a rate of 400 million keys per second per FPGA. Finally, we deliver estimates of the expected capabilities of the completely configured COPACOBANA with respect to the presented implementations.

2 Architectures for Cryptanalysis

Many algorithms tackling the most important problems in cryptanalysis can be implemented on FPGAs. However, code breaking involves more effort than just programming a single FPGA with a particular algorithm. Due to the enormous dimensions of cryptanalytical problems, much more resources than a single FPGA are required. What is needed is a powerful massively parallel machine, tweaked to the needs of the targeted algorithms.

Most problems can be parallelized and are perfectly suited for a distributed architecture. In our case, not much communication overhead is required. Conventional parallel computing architectures, such as provided by Cray, can in theory also be used for cryptanalytical applications. However, the cost-performance ratio is not optimized with this approach, resulting in prohibitively expensive attack machines. Similarly, many features of current high-end processors are not required for the targeted cryptanalytical problems. For instance, high-speed communication between CPUs, fast floating point operations, etc., cannot be used in our context. All of these features usually increase the cost of such a device, which is in particular annoying when they are superfluous. Even a simple grid of conventional PCs is not efficient, as can be seen from implementations of DES: An implementation on a single low-cost FPGA can be more than 100 times faster than an implementation on a conventional PC, while the FPGA is much cheaper than the PC. But certainly there is a tradeoff, e.g., a small key space might be manageable by PCs which are usually available in contrast to special purpose hardware which must be purchased or build in this case. For larger problems however, a custom design is inevitable in order to obtain a low-cost architecture with the required performance.

Our metric to decide whether an architecture is "good" or not is a function of performance, flexibility, and monetary cost. A good performance metric for hardware implementations is the area-time (AT) complexity. Whenever we can minimize the AT-complexity, the design can be called efficient. ASIC implementations can be AT-minimal and are the best choice for high-volume applications. However, ASICs are not flexible since they can implement only a single architecture. FPGAs in contrast are reprogrammable and, thus, are flexible. Moreover, if only a relatively small number of chips ($< 10\ 000$) is required, FPGAs are preferable since the production of ASICs is profitable only when targeting high volumes.

In the following, we describe an optimized architecture for cryptanalytical purposes and its implementation as custom-designed FPGA machine. The first prototype of COPACOBANA hosts 120 FPGAs and was produced for € 8,980, including material and manufacturing costs. The work at hand presents the first outcomes of the resulting prototype[3] of a hardware code cracker, built of low-cost FPGAs.

2.1 An Optimal Architecture to Break Ciphers

The targeted DES brute-force attack (see Section 3) has the following characteristic: First, the computational expensive operations are parallelizable. Secondly, single parallel instances do not need to communicate with each other. Thirdly, the overall communication overhead is low, driven by the fact that the computation phase heavily outweighs the data input and output

[3]Note that at the time of writing, all tests have been accomplished on a prototype with a single DIMM module since the remaining modules have not been available yet due to a production delay.

phases. In fact, all processes are computing most of the time, without any input or output. Communication is almost exclusively used for initialization and reporting of results. A central control instance for the communication can easily be accomplished by a conventional (low-cost) PC, connected to the instances by a simple interface. No high-speed communication interface is required. Forthly, a DES brute-force attack and the corresponding implementation call for very little memory. As a consequence, the available memory on contemporary low-cost FPGAs such as the Xilinx Spartan3 is sufficient.

By the use of low-cost FPGAs, we can build a cost-efficient reprogrammable and flexible architecture capable of hosting all targeted architectures.

2.2 Realization of COPACOBANA

Recapitulating, the Cost-Optimized Parallel Code Breaker (COPACOBANA) fitting our needs consists of many independent low-cost FPGAs, connected to a host-PC via a standard interface, e.g., USB or Ethernet. Furthermore, such a standard interface allows to easily extend a host-PC with more than one COPACOBANA device. The initialization of FPGAs, the control, and the accumulation of results is done by the host. All time-critical computations are done by the FPGAs, which realize the actual cryptanalytical architecture which is described in Section 3. Since the exhaustive key search demands for plenty of computing power, the targeted platform aggregates up to 120 FPGAs (Spartan3-1000). Building a system of such a dimension with commercially available FPGA boards is certainly feasible, but comes with a cost penality. Hence we decided to design, layout, and build our own hardware. We considered several different design options. Our cost-performance optimized design became only feasible by strictly restricting all functionality to those directly necessary for code breaking, and to make several design choices based on readily available components and interfaces.

2.2.1 Choice of the FPGA Type

We decided to pick a contemporary low-cost FPGA for the design, the Xilinx Spartan3-1000 FPGA (XC3S1000, speed grade -4, FT256 packaging). This comes with 1 million system gates, 17280 equivalent logic cells, 1920 Configurable Logic Blocks (CLBs) equivalent to 7680 slices, 120 Kbit Distributed RAM (DRAM), 432 Kbit Block RAM (BRAM), and 4 digital clock managers (DCMs) [25].The choice for this chip was derived by an evaluation of size and cost over several FPGA series and types. COPACOBANA is capable of holding up to 120 of such FPGAs.

2.2.2 Architectural Concept

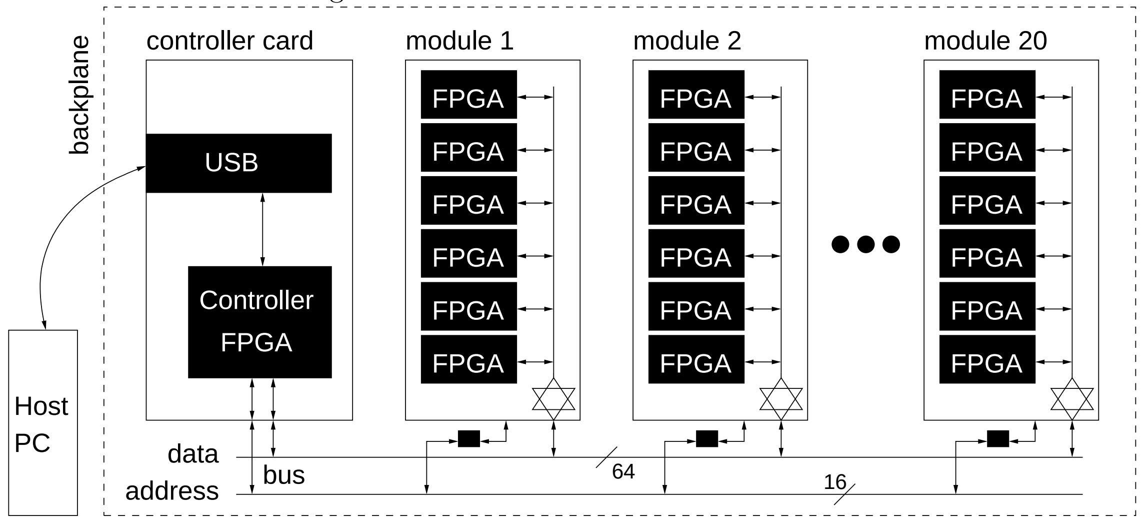
The design of COPACOBANA is depicted in Figure 1 and consists of

- *FPGA modules* for the actual implementation of the presented hardware architectures,
- a *backplane*, connecting all FPGA modules to a common data bus, address bus, and power supply,
- and a *controller card*, connecting the data bus and address bus to a host-PC via USB.

2.2.3 FPGA Modules

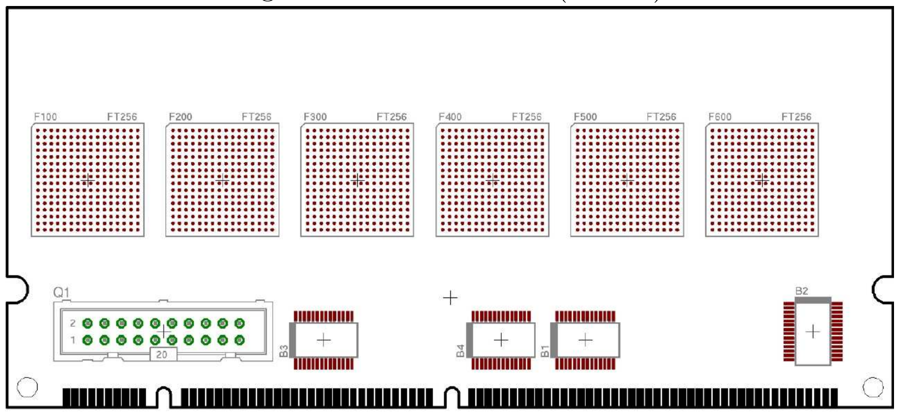
A step towards an extendable and simple architecture has been accomplished by the design of small pluggable FPGA modules. We decided to settle with small modules in the standard DIMM

Figure 1: Architecture of COPACOBANA



format, comprising 6 Xilinx XC3S1000 FPGAs. Figure 3 shows its realization as custom made 4-layer printed circuit board. The FPGAs are directly connected to a common 64-bit data bus on board of the FPGA module which is interfaced to the backplane data bus via transceivers with 3-state outputs.

Figure 2: FPGA module (DIMM)

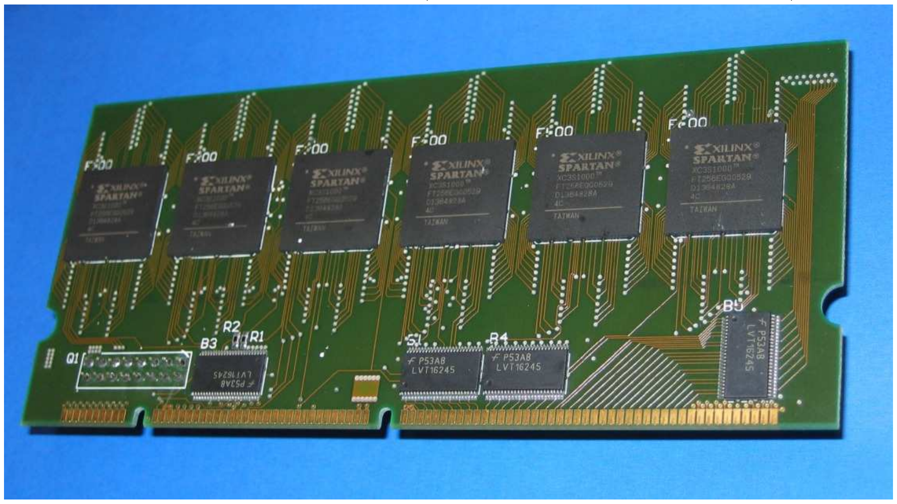


The DIMM format allows for a very compact component layout, which is important to closely connect the modules by a bus. Figure 2 depicts the chip arrangement. From the experience with current implementations on the same FPGA type, we dispense with active cooling of the FPGAs at these times. Depending on the heat dissipation of future applications, passive or active cooling might be an option for an upgrade.

2.2.4 Backplane

The backplane hosts all FPGA-modules and the controller card. All modules are connected by a 64-bit data bus and a 16-bit address bus. This single master bus is easy to control because no arbiter is required. Interrupt handling is totally avoided in order to keep the design as simple

Figure 3: FPGA module (4-layer printed circuit board)



as possible. If the communication scheduling of an application is unknown in advance, the bus master will need to poll the FPGAs.

Moreover, the power supply is routed to every FPGA module and the controller interface. The backplane distributes two clock signals from the controller card to the slots. Every FPGA module is assigned a unique hardware address, which is accomplished by Generic Array Logic (GAL) attached to every DIMM socket. Hence, all FPGA cores can have the same configuration and all FPGA modules can have the same layout. They can easily be replaced in case of a defect. Figure 4 shows the prototype of the backplane with a single FPGA module and the control interface card which will be described in the next subsection. The entire bus has been successfully tested by use of the prototype FPGA module with frequencies of up to 50 MHz. For the fully equipped board, the bus speed will be limited to 33 MHz due to power dissipation.

2.2.5 Control Interface

Data transfer from and to the FPGAs and to the host-PC is accomplished by the control interface. We decided to pick a small development board with an FPGA (CESYS USB2FPGA [2]) in favor of a flexible design. The development board comes with a Xilinx XC2S200 SPARTAN II FPGA (PQ208), an integrated USB 2.0 controller (CYPRESS FX-2), and 1 MByte SRAM. Moreover, the board provides an easy-pluggable 96-pin connector which we use for the connection to the backplane. In later versions of the design, it is also possible to replace the FPGA development board by a small microcontroller with a standard USB or Ethernet interface.

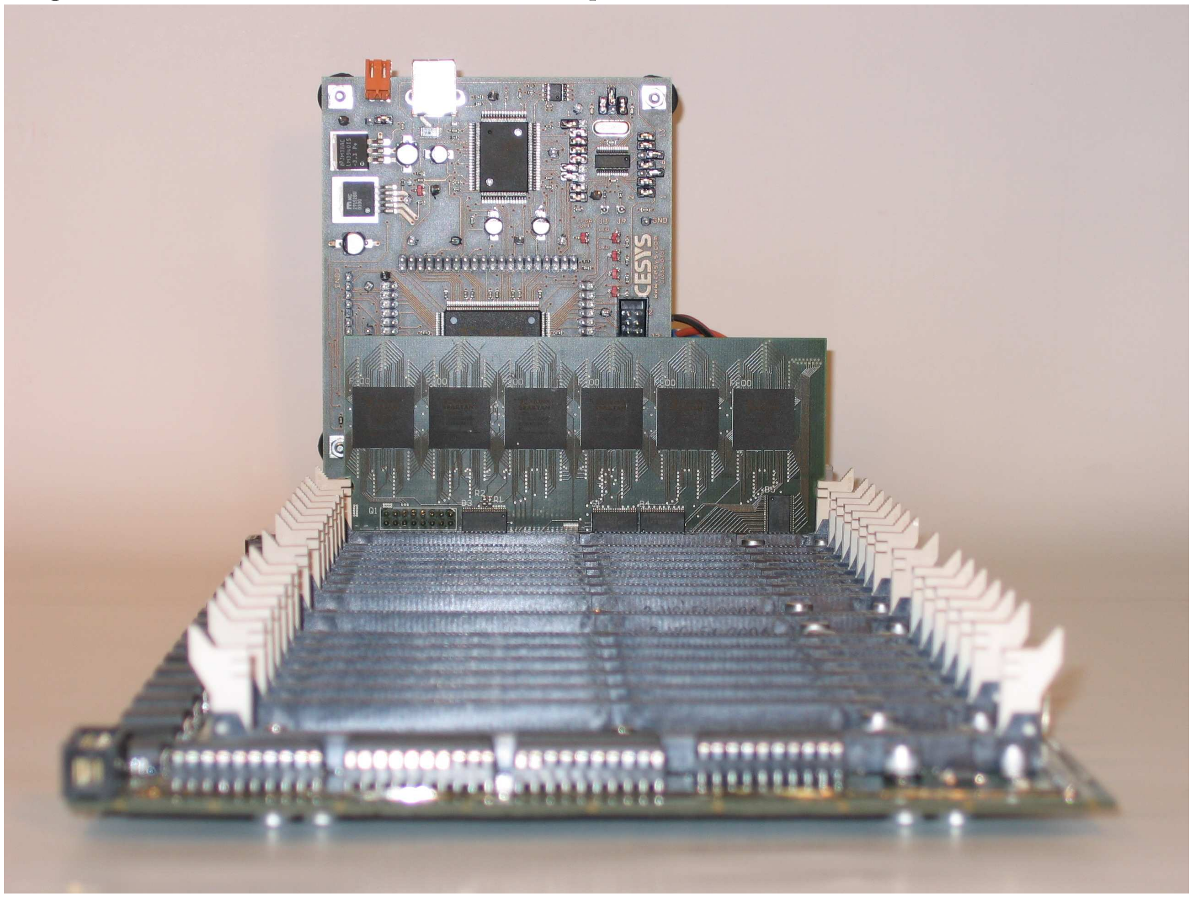
The controller hardware has to handle the adaptation of different clock rates: The USB interface uses a clock rate of 24 MHz, the backplane is clocked with 33 MHz, and the controller itself is running at an internal clock of 133 MHz. The internal clock is generated by an external clock synthesizer, the system clock is derived from a digital clock manager (DCM) present on the FPGA.

The main state machine of the control interface is used to

1. decode and execute host commands received via USB,

2. program the FPGAs via the data bus in slave parallel mode,

Figure 4: Modules of COPACOBANA: Backplane with FPGA module and controller card



3. initialize (write to) FPGAs and start the computation, and
4. regularly poll the FPGAs and check for new results.

Programming can be done for all FPGAs simultaneously, for a set of such, or for a particular one. Since the targeted cryptanalytic applications do not require different code on distinct FPGAs, a concurrent programming of all devices is very helpful. Details of the implementation of the control logic can be found in [20].

2.2.6 Host-PC

The top level entity of COPACOBANA is a host-PC which is used to program and control all FPGA implementations. For this purpose, a software library has been written to issue commands to the USB connected [20] controller card of COPACOBANA. All software routines are based on the closed source library provided by the board manufacturer (CESYS). With the low-level functions, FPGAs can be addressed and data can be stored and read to/ from a particular FPGA. Further functions include the detection of the hardware and some configuration routines such as, e.g., a backplane reset. Higher-level functions comprise commands at application level. E.g., for the DES Cracker, we can store a certain plaintext in the DES units, etc.

2.2.7 Hardware Cost

The total cost of the prototype of COPACOBANA amounts to € $8,980$ and is composed of the material and production costs of its components. Note that we did not include the NRE design costs for the layout and for programming the FPGAs. Hence the price of € $8,980$ are the actual third party costs we have for the fully extended COPACOBANA excluding NRE costs. Of course, in larger quantities further cost reductions should occur. Furthermore, the cost of the host-PC has not been included. However, a PC for approximately € 200 would be sufficient to control the exhaustive key search for COPACOBANA.

Table 1 lists the costs of the building blocks of COPACOBANA. For the FPGAs, we assume

Amount	**Type**	**Price p.p. in €**	**Total in €**
20	FPGA card (mounting included, price of FPGAs excluded)	100	2,000
20	Power card (for FPGA cards)	30	600
120	Xilinx XC3S1000 FPGA	40	4,800
1	Backplane	950	950
1	Controller board	590	590
1	Power supply (ATX)	40	40
Total			**8,980**

Table 1: Material and production cost of COPACOBANA

a price per piece of € 40 which, to our believe, reflects the actual price for the given quantity (> 100 pieces) but might differ depending on the point-of-sale and on the actual quantity[4]. E.g., for large volumes (> 250,000 pieces) Xilinx states that this particular type of FPGA costs less than € 10. Since this is the total cost of the prototype including some costly but unnecessary electronic parts, we assume a slightly lower price for the final version of COPACOBANA.

3 Breaking DES

In this section, we describe the implementation of a DES key search on the targeted low-cost FPGA, a Xilinx Spartan3-1000.

Ideally, the security of symmetric ciphers is dependent on the infeasibility of an exhaustive key search. This requires examining through each key in the possible key space. The cost of the 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 but making such brute force attacks impracticable.

The Data Encryption Standard (DES) with a 56-bit key size was chosen as the first commercial cryptographic standard by NIST in 1977 [11]. A key size of 56 bit was considered to be good choice considering the huge development costs for computing power in the late 70's, making a search over all the possible 2^{56} keys impractical. But DES has survived long beyond its recommended lifetime and still is being used in legacy systems or due to backward compatibility reasons. The advances in the hardware and decreasing costs have made DES vulnerable to brute force attacks.

[4]Depending on the quantity, prices for the XC3S1000 are in the range of € 10 to € 70.

3.1 Previous Work

There has 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 Whitfield Diffie and Martin Hellman [4] for a brute force machine that could find the key within a day at US$ 20 million, even though the authors later stated that this estimate had been too optimistic. The design proposal consisted of millions of specially designed VLSI chips working in parallel and each capable of searching one key in one microsecond.

A first detailed hardware design description for a brute force attacker was presented by Michael Wiener at the rump session of CRYPTO'93 and is reprinted in [23]. The machine could be built for less than a million US$ with $57,000$ DES chips that could recover a key every three and half hours. The estimates were updated in 1998 due to the advances in hardware for a million dollar machine to 35 minutes for each key recovery [24].

Ian Goldberg and David Wagner estimated the cost for building a DES brute force attacker using FPGAs at US$ 45,000 for a key recovery within a year [5]. In 1997, a detailed cost estimate for three different approaches for DES key search: Distributed computing, FPGAs, and custom ASIC designs, was compiled by a group of cryptographers [1].

The real practical attempts at breaking DES were encouraged by the RSA Secret Key challenge launched in 1997 [19]. The challenge was a 24 character known-plaintext attack.

The first challenge was solved by Rocke Verser, Matt Curtin, and Justin Dolske using the DESCHALL distributed network in 1997. The RSA DES Challenge II-1 was broken by *distributed.net* within 39 days in 1998. The RSA DES Challenge II-2 was won by the Electronic Frontier Foundation (EFF) DES hardware cracker called *Deep Crack* in 1998 within 56 hours [5]. The 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. The final blow to DES was given by the DES Challenge III which was solved in 22 hours 15 minutes using the combined effort of *Deep Crack* and *distributed.net*

A first low-cost approach in attacking a DES-based protocol was realized by [3]. The authors describe their experiences attacking the IBM 4758 CCA with an off-the-shelf FPGA development board.

Though this proved to be an end for DES for many applications, the huge cost involved to producing a machine like *Deep Crack* and access to foundries makes building such machines still impractical for smaller organizations. Therefore, we propose a more practical approach of an off-the-shelf-FPGA based hardware cracker.

3.2 DES on FPGAs

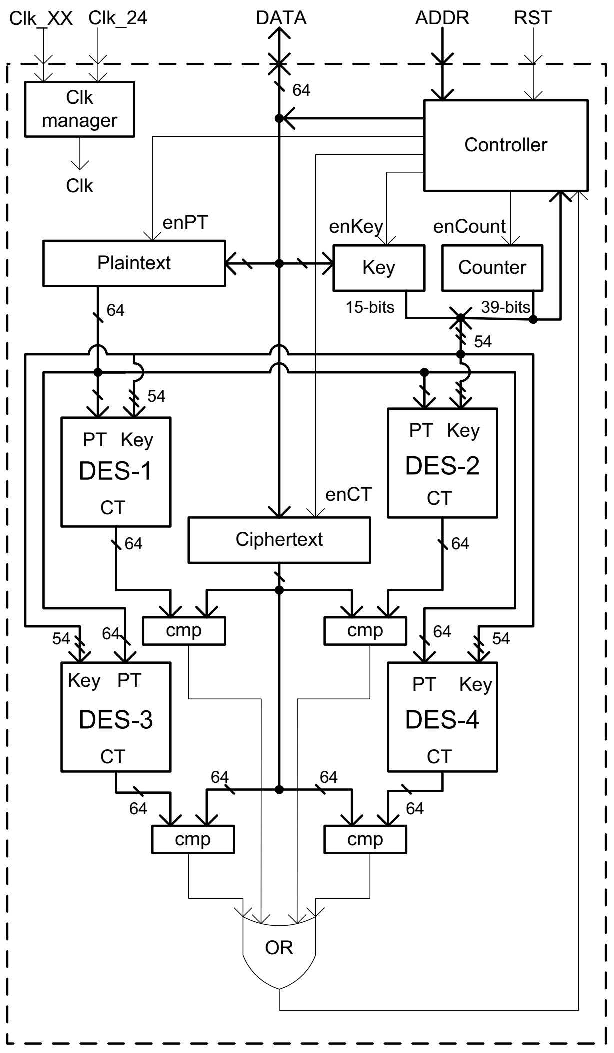
When DES was first proposed as a standard, its main application was seen in hardware based implementations. Hence DES is extremely efficient in terms of area and speed for hardware but unsuitable for a good software implementation due to the bit-level addressing in the design. Therefore an FPGA implementation of DES can be more than a 100 times faster than an implementation on a conventional PC at much lower costs. This allows a hardware based key search engine to be much faster and efficient compared to a software based approach.

3.2.1 The Key Search Engine and Subspace Search

The main aim of our key search engine is to check as many keys as possible in the least time to find the right key that could encrypt a known plaintext to its ciphertext that is made available. It is obvious that such a key search can be done in a highly parallelized fashion by partitioning the key space. This requires hardly any inter-process communication, as each of the DES engines can search for the right key within its allocated key subspace.

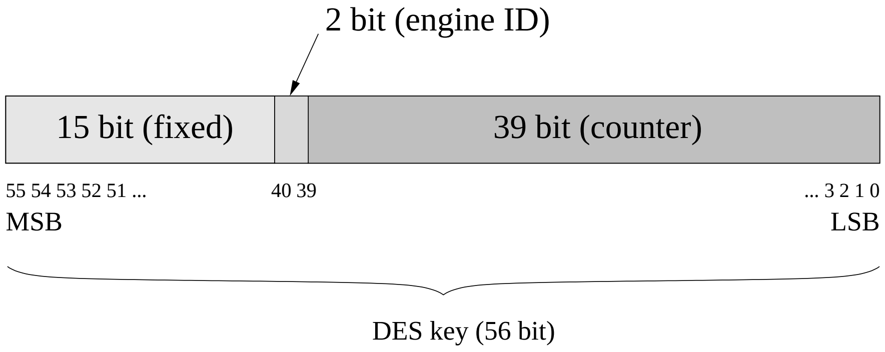
For the DES engine, we implemented a highly pipelined design of the Université Catholique de Louvain's Crypto Group [18]. A single DES engine can test one keys per clock and the pipelined architecture is adjusted such that the critical path is as small as possible, allowing for a fast implementation. With COPACOBANA, we can fit four such DES engines inside a single FPGA (Xilinx XC3S1000), and therefore allow for sharing of control circuitry and the key space. The design of the key search with the four DES engines inside an FPGA is as shown in Figure 5.

Figure 5: Overview of an FPGA with four DES key search units



It consists of a 64-bit *Plaintext* register and 64-bit *Ciphertext* register. The key space is allocated to each chip as the most-significant 15 bit of the key which is stored in the *Key* register. The *Counter* is used to run through the least significant 39 bit of the key. The remaining two bit of the 56-bit key for each of the DES engines is hardwired and is different for each of them. Thus, for every such FPGA, a task is assigned to search through all the keys with the 15 most-significant bit fixed, that is 2^{41} different keys. Figure 6 shows the partition of the key space.

Figure 6: Partition of the key space for one FPGA



The clock rate per FPGA is 100 MHz, hence, the key block can be searched in 5,500 seconds which is approximately 1.5 h. The estimated running time for a complete key search with COPACOBANA is discussed in Subsection 3.3.

3.2.2 Dividing the Key Space

The partitioning of the key space is done using a host-PC (described in Section 2.2.6). The key space is partitioned such that each chip takes around 90 minutes to check through its allocated key subspace, thus, avoiding huge communication requirements. This also allows the machine to restart the key search easily from a previous state if a power failure occurs. The generated cipher text (CT) is compared to that of the given *Ciphertext* stored in the register, using the comparator (*cmp*) block. The results of the four comparators are ORed and reported to the controller. If any of the DES engines provides a positive match, the controller reports the counter value to the host-PC. The host-PC keeps track of the key range that is assigned to each of the FPGAs and, hence, can match the right key from a given counter value. If no match is found until the counter overflows, the FPGA reports completion of the task and waits for a new key space to be assigned.

Since each FPGA can search through its key space totally independent of any other FPGA, only the host-PC needs to keep track of the number of FPGAs and the allocated key space. This kind of architecture for the key search engine makes it easily extendable, allowing more FPGAs to be added when required. The estimated time to complete the key search using COPACOBANA is discussed in the following.

3.3 Exhaustive Key Search with COPACOBANA

The running time for an average DES key search can easily be estimated based on the implemented architecture. Starting from the existing implementation, we can operate each of the FPGAs at 100 MHz and, therefore, each FPGA can check four keys every 10 ns. Consequently, a partial key space of 2^{41} keys can completely be checked in $2^{39} \cdot 10$ ns by a single FPGA, which is approximately 90 minutes. Since COPACOBANA will host 120 of these low-cost FPGAs, the key search machine will check $4 \cdot 120 = 480$ keys every 10 ns, i.e., 48 billion keys per second. To find the right key, COPACOBANA has to search through an average of 2^{55} different keys. Thus on average, COPACOBANA can find the right key after $(2^{55} \cdot 10)/480$ ns which is approximately

8.7 days. The time required for loading the plaintext, ciphertext and key space allocation are ignored as they are negligibly small compared to the overall running time.

3.4 A Case-Study: 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 one can choose from, a (cryptographically very weak) proprietary algorithm and the data encryption standard in CBC-Mode[5]. Parts of the interna and flaws of Diskreet we consider in the following, have also been reported in newsgroup postings by Gutmann [14] and Kocher [12].

3.4.1 DES-Key Generation

To encrypt a file or virtual disk, Diskreet asks for a password with a minimal length of 6 and a maximal length of 40 bytes. From this password a 64-bit DES-Key is generated, where as usual only 56 bit are actually used. The password-to-key mapping works as follows: First, leading whitespace characters are removed. Secondly, the C-code depicted below is executed which transforms lowercase to uppercase characters, divides the password into 8-byte blocks and computes the XOR-sum of these blocks.

```
unsigned char deskey[8] = { 0,0,0,0,0,0,0,0 };
for( i = 0; i < pwlength; i++ )
  {
    deskey[ i % 8 ] ^= toupper(password[ i ]);
  }
```

The resulting XOR-sum is used as DES-key. It is easy to see that this method of key generation is totally flawed, since the password-to-key mapping is not chaotic at all. Large “classes” of passwords are mapped to very restricted parts of the key space. More precisely, depending on the kind of characters of a password we obtain the following interesting subspaces of the DES-key space:

Key space α: Let us assume that all characters of a password are from the set {`A`, ..., `Z`, `@`, `[`, \, `]`, `^`, `_` }. 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. Table 2 shows the 16 possible key patterns. Since the least significant bit of each key byte is a parity bit, α contains in total $16 \cdot 2^{32} = 2^{36}$ DES-keys. If the password length is known a priori, the effective key length reduces to 32 bit.

Key space β: 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, β contains 2^{48} keys.

[5]More precisely, a slightly modified variant of DES is implemented, i.e., a fixed non-standard permutation is applied to all 8-byte blocks of the inputs and outputs (plaintext, ciphertext and DES-Key) of the DES-routines.

PW length mod 16	**DES key pattern ($64 = 8 \times 8$ bit)**
0	`000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx`
1	`010xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx`
2	`010xxxxx 010xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx`
3	`010xxxxx 010xxxxx 010xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx`
4	`010xxxxx 010xxxxx 010xxxxx 010xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx`
5	`010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 000xxxxx 000xxxxx 000xxxxx`
6	`010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 000xxxxx 000xxxxx`
7	`010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 000xxxxx`
8	`010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx`
9	`000xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx`
10	`000xxxxx 000xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx`
11	`000xxxxx 000xxxxx 000xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx`
12	`000xxxxx 000xxxxx 000xxxxx 000xxxxx 010xxxxx 010xxxxx 010xxxxx 010xxxxx`
13	`000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 010xxxxx 010xxxxx 010xxxxx`
14	`000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 010xxxxx 010xxxxx`
15	`000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 000xxxxx 010xxxxx`

Table 2: DES key patterns depending on password length modulo 16 for key space α

Key space γ: If we consider passwords consisting of arbitrary 8-bit ASCII characters then we obtain the whole DES-key space which contains 2^{56} different keys. We denote this key space by γ.

3.4.2 Password Check

Before performing a decryption, Diskreet first checks whether the correct password has been supplied. To enable this kind of verification process, Diskreet does the following additional steps prior to the actual encryption of user data: Diskreet puts the XOR-sum of the DES-key K destined for encryption and a 8-byte mask M in the header of the file which is used for storing the encrypted data. Then it encrypts the part of the header which 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.

Hence, in order to verify the correctness of a key K' one simply needs to test the following equality[6]

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

where C' denotes the 8-byte block of ciphertext which 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 3.4.1.

3.4.3 Key Search using COPACOBANA

An exhaustive key search for Norton Diskreet can easily be performed by a slight modification of the circuit depicted in Figure 5. Due to marginal changes, we still assume four DES cores on a single FPGA. The register for the plaintext contains the constant C from Equation 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

[6]Note that the equality check ignores the least significant bit of each byte.

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. Note that the DES core is pipelined and the particular key of the current output still is present in the last pipelining stage and does not require additional storage[7].

Depending on the actual key space (α, β, or γ), the counter output has to be connected to the four DES cores in different ways. In the case of key space γ, the key search can be adopted from Paragraph 3.2.1. The overall runtime of an average key search of 2^{55} keys does not change. With key space β, the fixed part of the key can be reduced to 7 bit, reducing the time of an average search of 2^{48} keys by a factor of 2^8 compared to key space γ. For the key space α, the fixed part of the key can be hard-wired and the counter can be reduced to 30 bit. In order to cover all possible key patterns (see Table 2) we can, e.g., use a simple 16-bit barrel shifter with the first 8 bit set to zero and the second 8-bit set to one. The first 8 bit are connected to the second bit in every byte of the key. With every counter overflow of the 30-bit counter, the barrel shifter performs a cyclic shift and, hence, changes the values of the key according to Table 2. We need 16 counter overflows to cover all possible password lengths modulo 16. In this case, the average search is reduced by a factor of 2^{20} compared to key space γ. If in addition the password length is known a priori, the barrel shifter is not required and the average search can be reduced by another factor of 2^4. Table 3 summarizes the number of DES operations and absolute timings required to break a Diskreet cipher with COPACOBANA.

Key space	**Remark**	**Average number of DES decryptions**	**Runtime**
α	Known password length	2^{31}	44.7 msec
α	Unknown password length	2^{35}	0.72 sec
β	7-bit ASCII	2^{47}	48.9 min
γ	8-bit ASCII	2^{55}	8.69 d

Table 3: Breaking Norton Diskreet with COPACOBANA

Obviously, a flawed key derivation function such as the one used in Norton Diskreet dramatically reduces the security of a cryptosystem. Compared to an average search time of 8.7 days for the complete key space γ, a key search for Norton Diskreet takes less than 50 ms in the fastest case, neglecting the setup phase of COPACOBANA for initializing the registers. Note that this cryptosystem allows for a maximum key space of size 2^{56} with an elaborate choice of the password. From a practical point of view however, the user has to face several obstacles regarding the input of passwords[8], preventing a simple input of secure passphrases.

Note that these results are only estimates based on a running implementation of the design as described in Section 3.2. However, both architectures are based on the same DES core and, thus, we do not expect a deviation from the estimates in case of an actual implementation.

[7]Due to the DES key schedule the key at that stage has to be shifted (permuted) before the XOR computation which can be accomplished by hardwiring.

[8]E.g., the input of ASCII symbols has to be done by typing the corresponding ASCII code, etc.

4 Further Cryptanalytical Applications

COPACOBANA can be deployed in various application scenarios. One important application concerns the current scheme for machine readable travel documents also known as ePass. In the following, we will show how such an architecture can be used to accelerate attacks on the ePass.

Beside a brute force attack against symmetric ciphers, we briefly sketch two further cryptanalytical applications which can efficiently be implemented on COPACOBANA:

- Factorization with the Elliptic Curve Method (ECM).
- Attacking ECC by solving discrete logarithms on elliptic curves with a parallel variant of Pollard's Rho method.

4.1 ePass

One important application of our architecture concerns the current scheme for machine readable travel documents, also known as ePass, which is initiated by organizations[9] in United States and several other countries to deploy biometric and RFID technologies for border and visa control. The claimed goal is to enhance security, protect against forgery and manipulation of travel documents and ease identity checks. The initiative has been subject to many political and technical debates. Several researchers have pointed out the security and privacy weaknesses of the deployed schemes and proposed improvements (see, e.g. [8, 9]). The cryptographic parts of the scheme shall consist of a Passive Authentication, Basic Access Control and an Active Authentication. Whereas Passive Authentication means that the data stored on an ePass are signed by the issuing nation, Basic Access Control should setup a secure (confidential) channel between the reader device (part of the inspection system) and the ePass chip and Active Authentication is deployed for anti-clonig purposes and requires an integer factorization based signature scheme implemented on the ePass chip. Note, that both Basic Access Control and Active Authentication are optional mechanisms. Basic Access Control is already implemented, e.g., in Germany and the Netherlands.

Current realizations of Basic Access Control deploy symmetric cryptography (Triple-DES) and generate the corresponding encryption and authentication keys from passport information that is visible in the physical document (e.g., serial number, date of birth and expiration date). More concretely, the key derivation scheme (e.g., implemented in reader devices) includes three computations of SHA-1, one to derive the chip individual key K_Seed, and two consecutive computations that derive encryption key K_Enc and authentication key K_MAC. One of the main concerns pointed out by many experts is the low entropy of this visible information being insecure for key generation. The scheme has been already successfully attacked using offline dictionary attacks[10].

Using our hardware architecture this kind of attack can be mounted in much shorter time, and even real-time, i.e., the time needed to pass the inspection system. Note that the dictionary attack can be accelerated by pre-computing possible encryption keys using SHA-1 in advance. Then our hardware only has to check for a matching of ciphertexts implementing Triple-DES only.

[9]More concretely, the International Civil Aviation Organization (ICAO).

[10]Experiments on the Netherlands' epass demonstrated that the encrypted information can be revealed in 2 hours after intercepting the communication, see `http://www.riscure.com/news/passport.html`. The issuing scheme in the Netherlands has about 35 bits of entropy.

Moreover, we are currently working on a device that can continuously read and record RF based communication at public places with high ePass density like airports. After the real-time decryption with our DES cracker, the information can be injected into distributed databases. Having installed such devices on many different airports and other similar places one can trace any person similar to tracing packages sent using postal services such as UPS.

4.2 Factorization

Since the introduction of public-key cryptography, the problem of factoring large composites is of increased interest. These days, the by far most popular asymmetric cryptosystem is RSA which was developed by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977 [17]. The security of the RSA cryptosystem relies on the difficulty of factoring large numbers. Hence, the development of a fast factorization method could allow for cryptanalysis of RSA messages and signatures. The best known method for factoring large integers is the General Number-Field Sieve (GNFS). One important step within the GNFS is the factorization of mid-size numbers for smoothness testing, an efficient algorithm for which is the Elliptic Curve Method (ECM). Since ECM is suitable for parallelization, it is promising to be implemented in hardware.

The algorithm itself is almost ideal for improving the area-time product through special purpose hardware. First, it performs a very high number of operations on a very small set of input data, and is, thus, not very I/O intensive. Secondly, it requires relatively little memory. Thirdly, the operands needed for supporting GNFS are well beyond the width of current computer buses, arithmetic units, and registers, so that special purpose hardware can provide a much better fit. This justifies the higher development costs compared to a solution with DSPs. Lastly, it should be noted that the nature of the application allows for a very high degree of parallelization.

The computation of ECM is a classical example for an algorithm that can be significantly accelerated through special-purpose hardware. The first reported implementation of ECM in hardware was used to factor numbers of up to 200 bit [13]. However, the monetary cost of the System-on-Chip hardware is quite high. Practical applications demand for a cheap realization of such ECM units. Therefore, a hardware platform consisting of many low-cost FPGAs seems to be an appropriate choice. As a result of the simple control logic for the ECM algorithm, no complex microcontroller is required and most logic can easily be put into the FPGA.

4.3 Elliptic Curve Discrete Logarithm

As we have seen in Section 4.2, the security of the popular asymmetric cryptosystem RSA is based on the difficulty of factorization. Besides factorization, many public-key cryptosystems are based on the difficulty of solving discrete logarithms in cyclic groups, known as the Discrete Logarithm Problem (DLP). A popular choice of such is the Elliptic Curve Cryptosystem (ECC) [7]. In contrast to the more efficient index-calculus attack on the DLP over finite fields on which, e.g., the Digital Signature Algorithm (DSA) is based, it is believed that ECC allows only for generic attacks such as the Pollard's rho (PR) method [16, 22]. This benefit yields much shorter underlying bit lengths for ECC (160...256 bit) compared to RSA or DLP in finite fields (1024...4096 bit) at an equivalent level of security [10].

Attacking ECC requires the same algorithmic primitives as the cryptosystem itself, namely point addition and point doubling. Similar to the case of ECM in the previous section, these primitives can be implemented very efficiently in hardware. A parallel PR algorithm is described in [22]. The first and, to our knowledge, only publication describing an actual hardware implementation of Pollard's Rho suitable for COPACOBANA is [6].

The PR algorithm essentially does a great many computations without the necessity of communication. Only particular results have to be reported to a central control unit, which can be realized by, e.g., a host-PC connected to the FPGA. The parameterization of the algorithm can be optimized for a low area-time product and a low communication overhead. Hence, the reports of the PR units to the host occur not very frequently.

5 Conclusion and Future Work

Cryptanalysis of symmetric and asymmetric ciphers is computationally extremely demanding. It is fair to believe that breaking codes with conventional PCs and super-computers is far too expensive. Bit-sizes of keys are chosen such that conventional methods of code breaking fail. The only promising way to tackle existing ciphers is to build special-purpose hardware, dedicated solely to suitable algorithms such as those presented in this paper. Conventional parallel architectures turn out to be far too complex and, thus, are not cost-efficient in solving cryptanalytical problems. Most of these problems can be parallelized easily and we show that the corresponding algorithms can be parameterized to lower the communication overhead. We present a cost-efficient hardware architecture (COPACOBANA) for less than € 10,000 (including material and production costs) which results from the algorithmic requirements of the targeted cryptanalytic problems.

The work at hand presents the design and first prototype of a cost-efficient design fulfilling the request. In its final extension, COPACOBANA will host 120 low-cost FPGAs. We showed, e.g., how the Data Encryption Standard (DES) can be broken within 9 days. The hardware design consists of reprogrammable logic and can be adopted to any suitable task, not necessarily restricted to code breaking.

We present an architecture and its implementation for an extremely efficient exhaustive key-search for the DES. With the implementation at hand, 48 billion keys can be tested per second with the COPACOBANA architecture, allowing for an average search time of less than 9 days. Even though DES has been replaced by the new Advanced Encryption Standard (AES), we believe that many existing cryptosystems still use DES for compatibility or legacy reasons. Hence, a low-cost DES cracker might finally give the incentive to replace DES. As shown in this paper, COPACOBANA can also be used for attacking password-based DES systems such as Norton Diskreet. Obviously, a flawed key derivation function such as the one used in Norton Diskreet can dramatically reduce the security of such a system. Compared to an average search time of 8.7 days for the complete key space, a smart key search for Norton Diskreet can take as little as 50 ms depending on the passphrase used as input to the key derivation function.

Besides an exhaustive key search, we furthermore sketched two possible applications for COPACOBANA, namely the elliptic curve method for factorization and Pollard's Rho for attacking ECC.

Future work includes completion and optimization of the brute-force attack for the fully configured COPACOBANA. The implementation has to be optimized to guarantee best possible throughput.

Recapitulating, COPACOBANA is the first and currently the only available low-cost design to solve cryptanalytical challenges. COPACOBANA was intended to, but is not necessarily restricted to solving problems related to cryptanalysis. Almost certainly there will exist more interesting problems apart from cryptology, which can be solved efficiently with the design at hand. In an ongoing project, we plan to apply the Smith-Waterman algorithm [26, 15] for scanning sequences of DNA or RNA against databases.

Acknowledgments

We like to thank the Xilinx Inc. for the generous donation of Spartan-3 FPGAs which formed the basis of our design. We are also indebted to Jean-Jacques Quisquater and François-Xavier Standaert of the Université Catholique de Louvain for making their high-speed DES design available. Furthermore, we would like to thank Kerstin Lemke and Ahmad-Reza Sadeghi for interesting discussions on the security of machine readable travel documents.

References

[1] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security: A Report by an Ad Hoc Group of Cryptographers and Computer Scientists. Technical report, January 1996. Available at `http://www.counterpane.com/keylength.html`.

[2] CESYS GmbH. USB2FPGA Product Overview. `http://www.cesys.com`, January 2005.

[3] R. Clayton and M. Bond. Experience Using a Low-Cost FPGA Design to Crack DES Keys. In B.S. Kaliski, C.K. Koc Cetin, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA*, volume 2523 of *series*, pages 579 – 592. Springer-Verlag, August 2002.

[4] W. Diffie and M. E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *COMPUTER*, 10(6):74–84, June 1977.

[5] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design.* O'Reilly & Associates Inc., July 1998.

[6] T.E. Güneysu. Efficient Hardware Architectures for Solving the Discrete Logarithm Problem on Elliptic Curves. Master's thesis, Horst Görtz Institute, Ruhr University of Bochum, February 2006.

[7] D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Verlag, 2004.

[8] A. Juels, D. Molnar, and D. Wagner. Security and privacy issues in e-passports. In *SecureComm 2005, First International Conference on Security and Privacy for Emerging Areas in Communication Networks, Athens, Greece*, September 2005.

[9] 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, April 2005.

[10] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[11] NIST FIPS PUB 46-3. *Data Encryption Standard.* Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, January 1977.

[12] Paul Kocher. Norton Diskreet (Security overview). posting to sci.crypt newsgroup, November 1993.

[13] J. Pelzl, M. Šimka, T. Kleinjung, J. Franke, C. Priplata, C. Stahlke, M. Drutarovský, V. Fischer, and C. Paar. Area-Time Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method. *IEE Proceedings Information Security*, 152(1):67–78, October 2005.

[14] Peter Gutmann. Norton's InDiskreet. posting to sci.crypt newsgroup, November 1993.

[15] G. Pfeiffer, H. Kreft, and M. Schimmler. Hardware Enhanced Biosequence Alignment. In *International Conference on METMBS*, pages 11–17. CSREA Press, 2005.

[16] J. M. Pollard. Monte carlo methods for index computation mod p. *Mathematics of Computation*, 32(143):918–924, July 1978.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[18] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES. In *Field-Programmable Logic and Applications - FPL*, pages 181–193, 2003.

[19] RSA Laboratories. Announcements: The RSA Data Security Secret-Key Challenge. *CRYPTOBYTES*, 2(3):16, 1997. Available at `ftp://ftp.rsa.com/pub/cryptobytes/crypto2n3.pdf`.

[20] C. Schleiffer. Design of a Host Interface for COPACOBANA. Technical report, Studienarbeit, Horst Görtz Institute, Ruhr University Bochum, January 2006.

[21] University of California, Berkeley. Seti@Home Website, 2005. `http://setiathome.berkeley.edu/`.

[22] P.C. van Oorschot and M.J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[23] M. J. Wiener. Efficient DES Key Search. In William R. Stallings, editor, *Practical Cryptography for Data Internetworks*, pages 31–79. IEEE Computer Society Press, 1996.

[24] M. J. Wiener. Efficient DES Key Search: An Update. *CRYPTOBYTES*, 3(2):6–8, Autumn 1997.

[25] Xilinx. Spartan-3 FPGA Family: Complete Data Sheet, DS099. `http://www.xilinx.com`, January 2005.

[26] C.W. Yu, K.H. Kwong, K.H. Lee, and P.H.W. Leong. A Smith-Waterman Systolic Cell. In *Proceedings of the 13th International Workshop on Field Programmable Logic and Applications — FPL 2003*, pages 375–384. Springer, 2003.