Security on FPGAs: State of the Art Implementations and Attacks

THOMAS WOLLINGER
Communication Security Group (COSY) - Ruhr-Universität Bochum
JORGE GUAJARDO
Infineon Technologies AG, Secure Mobile Solutions Division
and
CHRISTOF PAAR
Communication Security Group (COSY) - Ruhr-Universität Bochum

In the last decade, it has become aparent that embedded systems are integral parts of our every day lives. The wireless nature of many embedded applications as well as their omnipresence has made the need for security and privacy preserving mechanisms particularly important. Thus, as FPGAs become integral parts of embedded systems, it is imperative to consider their security as a whole. This contribution provides a state-of-the-art description of security issues on FPGAs, both from the system and implementation perspectives. We discuss the advantages of reconfigurable hardware for cryptographic applications, show potential security problems of FPGAs, and provide a list of open research problems. Moreover, we summarize both public and symmetric-key algorithm implementations on FPGAs.

Categories and Subject Descriptors: B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic—*Algorithms*; *Cost/performance*; B.7.1 [**Logic Design**]: Types and Design Styles—*Algorithms implemented in hardware*; *Gate arrays*; C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems; E.3 [**Data Encryption**]: Code breaking; Data encryption standard (DES); Public key cryptosystems

General Terms: Algorithms, Design, Performance, Security

Additional Key Words and Phrases: cryptography, security, attacks, reconfigurable hardware, FPGA, cryptographic applications, reverse engineering

This research was partially sponsored by the German Federal Office for Information Security (BSI).

Thomas Wollinger and Christof Paar are at the Department of Electrical Engineering and Information Sciences, Ruhr-Universität Bochum, Universitaetsstrasse 150, 44780 Bochum, Germany. Email: {wollinger,cpaar}@crypto.rub.de
Jorge Guajardo is at Infineon Technologies AG, Secure Mobile Solutions, St.-Martin Strasse 76, 81609 Munich, Germany. Email: Jorge.Guajardo@infineon.com
This work was partly done while the second author was at the COSY group.

1. MOTIVATION

Traditionally, in the design of embedded systems ASICs have been common components by providing the high performance and/or low power budget that many systems require at the expense of long and difficult design cycles. In the 1980s the use of reprogrammable components, in particular FPGAs, was introduced. FPGAs allowed for faster design cycles because they enabled early functionality testing. Nonetheless, the performance and size of FPGAs did not permit them to substitute ASICs in most applications and thus, they were mainly used to prototype embedded chips small enough to fit in the FPGA. In recent years, however, FPGAs manufacturers have come closer to filling the performance gap between FPGAs and ASICs, enabling them, not only to serve as fast prototyping tools but, also to become active players as components in embedded systems [Wong et al. 2002].

The trend in both industry (see for example [Altera Corporation 2000; 2002a; 2002b; Chameleon Systems Inc. ; Triscend Corporation ; Xilinx Inc. 2002; 2003]) and academia (see [Bondalapati and Prasanna 2002; Hauser and Wawrzynek 1997]) is to develop chips which include either embedded components in them such as memory, I/O controllers, and multiplier blocks, or both system reconfigurable components and programmable cores. The resulting processors/chips, which are not anymore a single part of an embedded system but rather can be used to develop the *whole* system, are known by various names ranging from hybrid architectures to Systems-on-Chip (SoC), Configurable System-on-Chip (CSoC), Reconfigurable Systems-on-Chip (RSoC), and Systems on Programmable Chip (SoPC), among others [Bondalapati and Prasanna 2002]. Thus, FPGAs and in particular reconfigurable devices are integral parts in embedded system design. This fact is exemplified by the great number of research publications in the area of FPGAs and applications such as image processing [Athanas and Abbott 1995], computer vision [Bondalapati and Prasanna 2002], solution of pattern recognition problems, e.g., text searching, fingerprinting matching, etc. [Buell et al. 1996], solution of boolean satisfiability problems [Rashid et al. 1998], digital signal processing [Tessier and Burleson 2000], and many others [Vuillemin et al. 1996]. For state-of-the-art surveys on reconfigurable computing and applications we refer the reader to [Bondalapati and Prasanna 2002; Compton and Hauck 2002; Schaumont et al. 2001]. However, to the authors' knowledge, there has not been a *comprehensive* treatment of security on FPGAs, both algorithmic and system issues, in the literature, even though there have been many articles looking at the implementation of a specific cryptographic algorithm on a specific FPGA over the last five years.

As FPGAs are becoming integral parts of embedded systems and many embedded applications require security mechanisms because of their very nature, it is imperative to consider security on FPGAs as a whole. As an example of this need, notice that a large share of all embedded applications are and will be wireless and will include sensing and actuation functions [Borriello and Want 2000], making the need for security and privacy preserving mechanisms obvious[1].

In actual cryptographic devices, FPGAs are rarely used as stand-alone component, especially SRAM-FPGAs. Very often, the FPGA will communicate with an

[1] In wireless applications the communication channel is easy to eavesdrop and to create profiles of persons, thus it is desirable to use cryptographic primitives [Davies and Gellersen 2002].

(embedded) microprocessor and/or an ASIC. In order to assure security for the over-all system, the other components and their interaction with the FPGA must also be considered. This is, however, beyond the scope of this contribution, which focuses on the security aspects of FPGA devices themselves. We would like to mention, that FPGAs are attractive for executing the actual cryptographic algorithms and are, thus, of particular importance from a security point of view.

1.1 Our Contributions

The reconfigurability of FPGAs offers major advantages when using them for cryptographic applications. Despite the vastness of the research literature on FPGA cryptographic implementations, there has been hardly any work regarding the suitability of FPGAs for security applications from a *systems* point of view. In particular, very little work has been done on the resistance of FPGAs to physical or system attacks, which, in practice, pose far a greater danger than algorithmic attacks. Thus, the first part of this paper is devoted to studying FPGAs from a systems security perspective. We do this by looking at attacks documented in the literature against FPGAs as well as attacks that have been performed against other hardware platforms and by adapting them and their solutions to FPGAs. Furthermore, we provide a list of open problems regarding system security of FPGAs.

As mentioned before, there has been a large amount of work done dealing with the algorithmic and computer architecture aspects of cryptographic schemes implemented on FPGAs over the last five years (see, e.g., relevant articles in [Koç and Paar 1999; 2000; Koç et al. 2001; Kaliski, Jr. et al. 2002]), often focusing on high-performance implementations. Nevertheless the works are scattered throughout the literature. Thus, the second part of this work, makes an attempt to organize the vast literature on FPGA cryptographic algorithm implementation according to the different hard mathematical problems on which the cryptographic primitives are based. In addition, taking performance (both area and throughput) as a measure of different implementations reported in the literature, we make recommendations as to which methods are best suited to implement cryptographic algorithms on FPGAs at the current state of technology. It should be noted that although we have tried to include the most up to date performance numbers available in the literature, some citations are a few years old. It can be assumed that due to the rapid progress in speed and integration density of commercial FPGAs, the speed and area numbers given in some of the references can be improved if implemented in state-of-the-art FPGAs. We would also like to stress that this work is not based on practical experiments, but on a careful analysis of available publications in the literature and on our experience implementing cryptographic algorithms. Given these facts, we hope that the contribution at hand is of interest to readers in academia, industry, and government sectors.

The remainder of this paper is organized as follows. Section 2 summarizes some of the advantages of FPGAs for cryptographic applications. Section 3 and 4 give an overview of the security shortcomings of FPGAs by sketching possible attacks and presenting possible countermeasures against the attacks. In Section 5 we discuss possible metrics to be used to compare the FPGA algorithm implementations that we survey in Sections 6 and Section 7. We end this contribution with a list of open problems regarding security applications in which FPGAs are used and some conclusions. We have also included a brief overview of cryptography in Appendix A as well as some of the characteristics of the most common FPGA families quoted in our literature survey in Appendix B.

2. SYSTEM ADVANTAGES OF FPGAS FOR CRYPTOGRAPHIC APPLICATIONS

In this section we list potential advantages of reconfigurable hardware (RCHW) in cryptographic applications. Some of the ideas are taken from our earlier work in [Elbirt et al. 2001] and further extended.

Algorithm Agility:. This term refers to the switching of cryptographic algorithms during operation of the targeted application. One can observe, that the majority of modern security protocols, such as SSL or IPsec, are algorithm independent and allow for multiple encryption algorithms. The encryption algorithm is negotiated on a per-session basis and a wide variety may be required; e.g., IPsec allows among others DES, 3DES, Blowfish, CAST, IDEA, RC4, and RC6 as algorithms, and future extensions are possible. Advantages of algorithm independent protocols are: 1) ability to delete broken algorithms 2) choose algorithms according to certain (e.g. personal) preferences 3) ability to add new algorithms. Whereas algorithm agility is costly with traditional hardware, FPGAs can be reprogrammed on-the-fly.

Algorithm Upload:. It is perceivable that fielded devices are upgraded with a new encryption algorithm. A reason for this could be that the product has to be compatible to new applications. From a cryptographical point of view, algorithm upload can be necessary because a current algorithm was broken (e.g., Data Encryption Standard - DES [Federal Information Processing Standards 1977]), a standard expired (e.g. DES), a new standard was created (e.g. Advanced Encryption Standard - AES [U.S. Department of Commerce/National Institute of Standard and Technology 2001a]), and/or that the list of ciphers in an algorithm independent protocol was extended. Assuming there is some kind of (temporary) connection to a network such as the Internet, FPGA-equipped encryption devices can upload the new configuration code. Notice that the upgrade of ASIC-implemented algorithms is practically infeasible if many devices are affected or if the systems are not easily accessible, for instance in satellites.

Architecture Efficiency:. In certain cases a hardware architecture can be much more efficient if it is designed for a specific set of parameters. Parameters for cryptographic algorithms can be for example the key, the underlying finite field, the coefficient used (e.g., the specific curve of an ECC system), and so on. Generally speaking, the more specific an algorithm is implemented the more efficient it can become. An efficient parameter-specific implementation of the symmetric cipher IDEA [Lai and Massey 1990; Lai et al. 1991] was presented in [Taylor and Goldstein 1999]. With fixed keys, the main operation in the IDEA degenerates into a constant multiplication which is far more efficient than a general modular multiplication. In the case of IDEA a general modular shift-and-add multiplication requires 16 partial multiplications and only eight for a fixed key. Another example taken from asymmetric cryptography is the arithmetic architectures for Galois fields. These architectures tend to be (far) more efficient if the field order and irreducible polynomial are fixed. Squaring in $GF(2^m)$ takes $m/2$ cycles with a general architecture, but only one cycle if the architecture is compiled for a fixed field [Wu

1999]. Notice that squaring in $GF(2^m)$ is one of the most common operations when implementing elliptic curve cryptosystems defined over fields of characteristic two. FPGAs allow this type of design and optimization with specific parameter set. Due to the nature of FPGAs, the application can be changed totally or partially.

Resource Efficiency:. The majority of security protocols are hybrid protocols, e.g. IPSec [Kent and Atkinson 1998], SSL [Freier et al. 1996], TLS [Dierks and Allen 1999]. This implies, that a public-key algorithm is used to transmit the session key. After the key was established a private-key algorithm is needed for data encryption. Since the algorithms are not used simultaneously, the same FPGA device can be used for both through run-time reconfiguration.

Algorithm Modification:. There are applications which require modification of standardized cryptographic algorithms, e.g., by using proprietary S-boxes or permutations. Such modifications are easily made with RCHW. One example, where a standardized algorithm was slightly changed, is the UNIX password encryption [Menezes et al. 1997] where DES is used 25 times in a row and a 12-bit salt modifies the expansion mapping (the Unix password encryption has been standardized [ANSI 1981]). It is also attractive to customize block cipher such as DES or AES with proprietary S-boxes for certain applications. Furthermore, in many occasions cryptographic primitives or their modes of operation have to be modified according to the application.

Throughput:. General-purpose CPUs are not optimized for fast execution especially in the case of public-key algorithms. Mainly because they lack instructions for modular arithmetic operations on long operands. Modular arithmetic operations include for example exponentiation for RSA [Rivest et al. 1978] and multiplication, squaring, inversion, and addition for elliptic curve cryptosystems (ECC) [Koblitz 1987; Miller 1986a]. Although typically slower than ASIC implementations, FPGA implementations have the potential of running substantially faster than software implementations. The block cipher AES, exemplarily, reaches a data rate of 112,3 Mbit/s and 718,4 Mbit/s on a DSP TI TMS320C6201 [Wollinger et al. 2000] and Pentium III [Lipmaa 2002], respectively. In comparison, the FPGA implementation of the same algorithm on a Virtex XCV-1000BG560-6 achieved 12 GBit/s using 12,600 slices and 80 RAMs [Gaj and Chodowiec 2001]. On the other hand, an ASIC encrypts at about double the speed of the FPGA, e.g. Amphion CS5240TK reaches 25.6 Gbit/s at 200MHz [Amphion].

Cost Efficiency:. There are two cost factors, that have to be taken into consideration, when analyzing the cost efficiency of FPGAs: cost of development and unit prices. The costs to develop an FPGA implementation of a given algorithm are much lower than for an ASIC implementation, because one is actually able to use the given structure of the FPGA (e.g. look-up table) and one can test the reconfigured chip endless times without any further costs. This results in a shorter time-to-market period, which is nowadays an important cost factor. The unit prices are not so significant when comparing them with the development costs. However, for high-volume applications, ASIC solutions usually become the more cost-efficient choice.

We would like to stress that the advantages above are not necessarily restricted to cryptographic applications. They can also be exploited in different contexts.

For example, the remote upload of a configuration can be used to fix bugs in fielded devices, or to upgrade existing devices to make them compatible with new standards. However, some of the cryptographic advantages, such as replacement of a broken algorithm, may carry more weight for a system design than advantages in different application domains.

Note that the listed potential advantages of FPGAs for cryptographic applications can only be exploited if the following questions pertaining to the security properties of FPGAs have been addressed:

—Are there any shortcomings in terms of security using FPGAs (e.g. reverse-engineering of the bitstream)?

—Are physical (active) attacks possible? If so, how much effort is involved compared to attacking an ASIC?

—Do manufactures add features that may lower the security of the FPGA?

—What kind of known attacks can be applied to FPGA implementations?

3. SECURITY SHORTCOMINGS OF FPGAS

This section summarizes security problems produced by attacks against given FPGA implementations. First we would like to state what the possible goals of such attacks are.

3.1 Objectives of an Attacker

The most common threat against an implementation of a cryptographic algorithm is to learn a confidential cryptographic key, that is, either a symmetric key or the private key of an asymmetric algorithm. Given that the algorithms applied are publicly known in most commercial applications, knowledge of the key enables the attacker to decrypt future (assuming the attack has not been detected and countermeasures have not been taken) and, often more harming, past communications which had been encrypted. Another threat is the one-to-one copy, or "cloning", of a cryptographic algorithm *together* with its key. In some cases it can be enough to run the cloned application in decryption mode to decipher past and future communications. In other cases, execution of a certain cryptographic operation with a presumingly secret key is in most applications the sole criteria which authenticates a communication party. An attacker who can perform the same function can masquerade as the attacked communication party. Yet another threat is given in applications where the cryptographic algorithms are proprietary. Even though such an approach is not wide-spread, it is standard practice in applications such as pay-TV and in government communications. In such scenarios it is already interesting for an attacker to reverse-engineer the encryption algorithm itself. The associated key might later be recovered by other methods (e.g., bribery or classical cryptanalysis.) The discussion above assumes mostly that an attacker has physical access to the encryption device. Whether that is the case or not depends heavily on the application. However, we believe that in many scenarios such access can be assumed, either through outsiders or through dishonest insiders.

In the following we discuss vulnerabilities of modern FPGAs against such attacks. In areas were no attacks on FPGAs have been published, we tried to extrapolate from attacks on other hardware platforms, mainly memory cell and chip cards.

3.2 Black Box Attack

The classical method to reverse engineer a chip is the so called Black Box attack. The attacker inputs all possible combinations, while saving the corresponding outputs. The intruder is then able to extract the inner logic of the FPGA, with the help of the Karnaugh map or algorithms that simplify the resulting tables. This attack is only feasible if a small FPGA with explicit inputs and outputs is attacked and a lot of processor power is available. The reverse engineering effort grows and it will become less feasible as the size and complexity of the FPGA increases. The cost of the attack, furthermore, rises with the usage of state machines, LFSRs (Linear Feedback Shift Registers), integrated storage, and, if pins can be used, input and output [Dipert 2000].

3.3 Readback Attack

Readback is a feature that is provided for most FPGA families. This feature allows to read a configuration out of the FPGA for easy debugging. An overview of the attack is given in [Dipert 2000]. The idea of the attack is to read the configuration of the FPGA through the JTAG or programming interface in order to obtain secret information (e.g. keys, proprietary algorithm). The readback functionality can be prevented with a security bit. In some FPGA families, more than one bit is used to disable different features, e.g., the JTAG boundary. In [Aplan et al. 1999], the idea of using a security antifuse to prevent readout of information is patented.

However, it is conceivable, that an attacker can overcome these countermeasures in FPGA with fault injection. This kind of attack was first introduced in [Boneh et al. 1997]. The authors showed how to break public-key algorithms, such as the RSA and Rabin signature schemes, by exploiting hardware faults. Furthermore, they give a high level description of transient faults, latent faults, and induced faults. This publication, was followed by [Biham and Shamir 1997], where the authors introduced differential fault analysis, which can potentially be applied against all symmetric algorithms in the open literature. Meanwhile there have been many publications that show different techniques to insert faults, e.g., electro magnetic radiation [Quisquater and Samyde 2001], infrared laser [Ajluni 1995], or even a flash light [Skorobogatov and Anderson 2002]. It seems very likely that these attacks can be easily applied to FPGAs, since they are not especially targeted to ASICs. Therefore, one is able to deactivate security bits and/or the countermeasures, resulting in the ability to read out the configuration of the FPGA [Kessner 2000; Dipert 2000].

Despite these attacks Actel Corporation [Actel Corporation 2002] claims that after the programming phase, the cells of FPGAs cannot be read at all. On the other hand Xilinx offers the users the software tool JBits [Guccione and Levi], which provides an API to access the bitstream information and allows dynamic reconfiguration for Xilinx Virtex FPGAs. JBits allows a simplified and automated access to specific part of the bitstream, resulting in a extra advantage for the attacker who performs a readback attack.

3.4 Cloning of SRAM FPGAs

The security implications that arise in a system that uses SRAM FPGAs are obvious, if the configuration data is stored unprotected in the system but external to the FPGA. In a standard scenario, the configuration data is stored externally in nonvolatile memory (e.g., PROM) and is transmitted to the FPGA at power up in order to configure the FPGA. An attacker could easily eavesdrop on the transmission and get the configuration file. This attack is therefore feasible for large organizations as well as for those with low budgets and modest sophistication.

3.5 Reverse-Engineering of the Bitstreams

The attacks described so far output the bitstream of the FPGA design. In order to get the design of proprietary algorithms or the secret keys, one has to reverse-engineer the bitstream. The condition to launch the attack is not only that the attacker has to be in possession of the bitstream, but furthermore the bitstream has to be in the clear, meaning it is not encrypted.

FPGA manufactures claim, that the security of the bitstream relies on the disclosure of the layout of the configuration data. This information will only be made available if a non-disclosure agreement is signed, which is, from a cryptographic point of view, an extremely insecure situation. This security-by-obscurity approach was broken at least ten years ago when the CAD software company NEOCad reverse-engineered a Xilinx FPGA. NEOCad was able to reconstruct the necessary information about look-up tables, connections, and storage elements [Seamann 2000]. Hence, NEOCad was able to produce design software without signing non-disclosure agreements with the FPGA manufacturer. Even though a big effort has to be made to reverse engineer the bitstream, for large organizations it is quite feasible. In terms of government organizations as attackers, it is also possible that they will get the information of the design methodology directly from the vendors or companies that signed NDAs.

3.6 Physical Attack

The aim of a physical attack is to investigate the chip design in order to get information about proprietary algorithms or to determine the secret keys by probing points inside the chip. Hence, this attack targets parts of the FPGA, which are not available through the normal I/O pins. This can potentially be achieved through visual inspections and by using tools such as optical microscopes and mechanical probes. However, FPGAs are becoming so complex that only with advanced methods, such as Focused Ion Beam (FIB) systems, one can launch such an attack. To our knowledge, there are no countermeasures to protect FPGAs against this form of physical threat. In the following, we will try to analyze the effort needed to physically attack FPGAs manufactured with different underlying technologies.

3.6.1 *SRAM FPGAs.* Unfortunately, there are no publications available that accomplished a physical attack against SRAM FPGAs. This kind of attack is only treated very superficially in a few articles, e.g. [Richard 1998]. In the related area of SRAM memory, however there has been a lot of effort by academia and industry to exploit this kind of attack [Gutmann 1996; 2001; Anderson and Kuhn 1997; Williams et al. 1996; Schroder 1998; Soden and Anderson 1993; Kommerling and Kuhn 1999]. Due to the similarities in structure of the SRAM memory cell and the internal structure of the SRAM FPGA, it is most likely that the attacks can be employed in this setting.

Contrary to common wisdom, the SRAM memory cells do not entirely loose the contents when power is cut. The reason for these effects are rooted in the physical properties of semiconductors (see [Gutmann 2001] for more details). The physical changes are caused mainly by three effects: electromigration, hot carriers, and ionic contamination.

In the published literature one can find several different techniques to determine the changes in device operations. Most publications agree that device can be altered, if 1) threshold voltage has changed by 100mV or 2) there is a 10% change in transconductance, voltage or current. An extreme case of data recovery, was described in [Anderson and Kuhn 1997]. The authors were able to extract a DES master key from a module used by a bank, without any special techniques or equipment on power-up. The reason being that the key was stored in same SRAM cells over a long period of time. Hence, the key was "burned" into the memory cells and the key values were retained even after switching off the device.

"I_{DDQ} testing" is one of the widely used methods and it is based on the analysis of the current usage of the device. The idea is to execute a set of test vectors until a given location is reached, at which point the device current is measured. Hot carrier effects, cell charge, and transitions between different states can then be detected at the abnormal I_{DDQ} characteristic [Gutmann 2001; Williams et al. 1996]. In [Schroder 1998], the authors use the substrate current, the gate current, and the current in the drain-substrate diode of a MOSFET to determine the level and duration of stress applied.

When it becomes necessary to access internal portions of a device, there are also alternative techniques available to do so, as described in [Soden and Anderson 1993]. Possibilities are to use the scan path that the IC manufacturers insert for test purposes or techniques like bond pad probing [Gutmann 2001]. When it becomes necessary to use access points that are not provided by the manufacturer, the layers of the chip have to be removed. Mechanical probing with tungsten wire with a radius of $0,1 - 0,2\mu m$ is the traditional way to discover the needed information. These probes provide gigahertz bandwidth with $100fF$ capacitance and $1M\Omega$ resistance. Due to the complex structure and the multi layer production of chips the mechanical testing is not sufficient enough. Focused Ion Beam (FIB) workstations can expose buried conductors and deposit new probe points. The functionality is similar to an electron microscope and one can inspect structures down to 5nm [Kommerling and Kuhn 1999]. Electron-beam tester (EBT) is another measurement method. An EBT is a special electron microscope that is able to speed primary electrons up to 2.5 kV at 5nA. EBT measures the energy and amount of secondary electrons that are reflected.

Resulting from the above discussion of attacks against SRAM memory cells, it seems likely that a physical attack against SRAM FPGAs can be launched successfully, assuming that the described techniques can be transfered. However, the physical attacks are quite costly and having the structure and the size of state-of-the-art FPGA in mind, the attack will probably only be possible for large organizations, for example intelligence services.

3.6.2 *Antifuse FPGAs.* To discuss physical attacks against antifuse (AF) FPGAs, one has to first understand the programming process and the structure of the cells. The basic structure of an AF node is a thin insulating layer (smaller than $1\mu m^2$) between conductors that are programmed by applying a voltage. After applying the voltage, the insulator becomes a low-resistance conductor and there exists a connection (diameter about 100nm) between the conductors. The programming function is permanent and the low-impedance state will persist indefinitely.

In order to be able to detect the existence or non-existence of the connection one has to remove layer after layer, or/and use cross-sectioning. Unfortunately, no details have been published regarding this type of attack. In [Dipert 2000], the author states that a lot of trial-and-error is necessary to find the configuration of one cell and that it is likely that the rest of the chip will be destroyed, while analyzing one cell. The main problem with this analysis is that the isolation layer is much smaller than the whole AF cell. One study estimates that about 800,000 chips with the same configuration are necessary to explore the configuration file of an Actel A54SX16 chip with 24,000 system gates [Dipert 2000]. Another aggravation of the attack is that only about 2–5 % of all possible connections in an average design are actually used. In [Richard 1998] a practical attack against AF FPGAs was performed and it was possible to alter one cell in two months at a cost of $1000. Based on these arguments some experts argue that physical attacks against AF FPGAs are harder to perform than against ASICs [Actel Corporation 2002]. On the other hand, we know that AF FPGAs can be easily attacked if not connected to a power source. Hence, it is easier to drill holes to disconnect two connections or to repair destroyed layers. Also, depending on the source, the estimated cost of an attack and its complexity are lower [Richard 1998].

3.6.3 *Flash FPGAs.* The connections in flash FPGAs are realized through flash transistors. That means the amount of electrons flowing through the gate changes after configuration and there are no optical differences as in the case of AF FPGAs. Thus, physical attacks performed via analysis of the FPGA cell material are not possible. However, flash FPGAs can be analyzed by placing the chip in a vacuum chamber and powering it up. The attacker can then use a secondary electron microscope to detect and display emissions. The attacker has to get access to the silicon die, by removing the packet, before he can start the attack [Dipert 2000]. However, experts are not certain about the complexity of such an attack and there is some controversy regarding its practicality [Actel Corporation 2002; Richard 1998]

Other possible attacks against flash FPGAs can be found in the related area of flash memory. The number of write/erase cycles are limited to 10,000 – 100,000, because of the accumulation of electrons in the floating gate causing a gradual rise of the transistors threshold voltage. This fact increases the programming time and eventually disables the erasing of the cell [Gutmann 2001]. Another less common failure is the programming disturbance in which unselected erased cells gain charge when adjacent selected cells are written [Aritome et al. 1993]. This failure does not change the read operations but it can be detected with special techniques described in [Gutmann 2001]. Furthermore, there are long term retention issues, like electron emission. The electrons in the floating gate migrate to the interface with the underlying oxide from where they tunnel into the substrate. This emission causes a net charge loss. The opposite occurs with erased cells where electrons are injected [Papadas et al. 1991]. Ionic contamination takes place as well but the influence on the physical behavior is so small that it can not be measured. In addition, hot carrier effects have a high influence, by building a tunnel between the bands. This causes a change in the threshold voltage of erased cells and it is especially significant for virgin cells [Haddad et al. 1989]. Another phenomenon is overerasing, where an erase cycle is applied to an already-erased cell leaving the floating gate positively charged. Thus, turning the memory transistor into a depletion-mode transistor [Gutmann 2001].

All the described effects change in a more or less extensive way the cell threshold voltage, gate voltage, or the characteristic of the cell. We remark that the stated phenomenons apply for EEPROM memory and that due to the structure of the FPGA cell these attacks can be simply adapted to attack flash/EEPROM FPGAs.

3.6.4 *Conclusions on Physical Attacks.* It is our position that due to the lack of published physical attacks against FPGAs, it is very hard (if at all possible) to predict the costs of such an attack. It is even more difficult to compare the effort needed for such an attack to a similar attack against an ASIC as there is no publicly available contribution which describes a physical attack against an FPGA that was completely carried out. Nevertheless, it is possible to draw some conclusions from our above discussion.

First, we notice that given the current size of state-of-the-art FPGAs, it seems unfeasible, except perhaps for large government organizations and inteligent services, to capture the whole bitstream of an FPGA. Having said that, we should caution that in some cases, an attacker might not need to recover the whole bitstream information but rather a tiny part of it, e.g., the secret-key. This is enough to break the system from a practical point of view and it might be feasible.

On the other hand, there are certain properties that *might* increase the effort required for a physical attack against FPGAs when compared to ASICs. In the case of SRAM-, Flash-, EPROM-, and EEPROM-FPGAs there is no printed circuit (as in the case of ASICs) and therefore it is potentially harder to find the configuration of the FPGA. The attacker has to look for characteristics that were changed on the physical level during the programming phase. In the case of antifuse FPGAs the effort for a physical attack *might* increase compared to ASICs because one has to find a tiny connection with a diameter of about 100 nm in a 1 μm^2 insulation layer. Furthermore, only $2-5\%$ of all possible connections are used in an average configuration.

3.7 Side channel attacks

Any physical implementation of a cryptographic system might provide a *side channel* that leaks unwanted information. Examples for side channels include in particular: power consumption, timing behavior, and electromagnet radiation. Obviously, FPGA implementations are also vulnerable to these attacks. In [Kocher et al. 1999] two practical attacks, Simple Power Analysis (SPA) and Differential Power Analysis (DPA) were introduced. The power consumption of the device while performing a cryptographic operation was analyzed in order to find the secret keys from a tamper resistant device. The main idea of DPA is to detect regions in the power consumption of a device which are correlated with the secret key. Moreover, in some cases little or no information about the target implementation is required. Since their introduction, there has been a lot of work improving the original power attacks (see, e.g., relevant articles in [Koç and Paar 1999; 2000; Koç et al. 2001; Kaliski, Jr. et al. 2002]). More recently the first successful attacks based on the analysis of electromagnetic emissions have also been published [Agrawal et al. 2002]. Even though most of the published attacks are not specific to a particular platform, there has usually been an assumption that the underlying platform is either software or an ASIC. There seems to be very little work at the time of writing addressing the feasibility of actual side channel attacks against FPGAs. Very recently the first experimental results of simple power analysis on an ECC implementation on an FPGA have been presented in [Örs et al. 2003] and on RSA and DES implementations in [Standaert et al. 2003]. Somewhat related was the work presented in [Shang et al. 2002] which concludes that 60% of the power consumption in a XILINX Virtex-II FPGA is due to the interconnects and 14% and 16% is due to clocking and logic, respectively. These figures would seem to imply that and SPA type attack would be harder to implement on an FPGA than on an ASIC. However, the results presented in [Standaert et al. 2003; Örs et al. 2003] show that SPA attacks are feasible on FPGAs and that they can be realized in practice.

4. HOW TO PREVENT THE POSSIBLE ATTACKS?

This section shortly summarizes possible countermeasures that can be provided to minimize the effects of the attacks mentioned in the previous section. Most of them have to be realized by design changes through the FPGA manufacturers, but some could be applied during the programming phase of the FPGA.

4.1 Preventing the Black Box Attack

The Black Box Attack is not a real threat nowadays, due to the complexity of the designs and the size of state-of-the-art FPGAs (see Section 3.2). Furthermore, the nature of cryptographic algorithms prevents the attack as well. Cryptographic algorithms can be segmented in two groups: symmetric-key and public-key algorithms. Symmetric-key algorithms can be further divided into stream and block ciphers. Today's stream ciphers output a bit stream, with a period length of 128 bits [Thomas et al. 2003]. Block ciphers, like AES, are designed with a block length of 128 bits and a minimum key length of 128 bits. Minimum length in the case of public-key algorithms is 160 bits for ECC and 1024 bits for discrete logarithm and RSA-based systems. It is widely believed, that it is infeasible to perform a brute force attack and search a space with 2^{80} possibilities. Hence, implementations of this algorithms can not be attacked with the black box approach.

4.2 Preventing the Cloning of SRAM FPGAs

There are many suggestions to prevent the cloning of SRAM FPGAs, mainly motivated by the desire to prevent reverse engineering of general, i.e., non-cryptographic, FPGA designs. One solution would be to check the serial number before executing the design and delete the circuit if it is not correct. This approach is not practical because of the following reasons: 1) The whole chip, including the serial number can be easily copied; 2) Every board would need a different configuration; 3) Logistic complexity to manage the serial numbers [Kessner 2000]. Another solution would be to use dongles to protect the design [Kean 2001; Kessner 2000]. Dongles are based on security-by-obscurity, and therefore do not provide solid security, as it can be seen from the software industry's experience using dongles for their tools. A more realistic solution would be to have the nonvolatile memory and the FPGA in one chip or to combine both parts by covering them with epoxy. This reflects also the trend in chip manufacturing to have different components combined, e.g., the FPSLIC from Atmel. However, it has to be guaranteed that an attacker is not able to separate the parts.

Encryption of the configuration file is the most effective and practical countermeasure against the cloning of SRAM FPGAs. There are several patents that propose different scenarios related to the encryption of the configuration file: how to encrypt, how to load the file into the FPGA, how to provide key management, how to configure the encryption algorithms, and how to store the secret data [Jeffrey 2002; Austin 1995; Erickson 1999; Sung and Wang 1999; Algotronix Ltd.]. In [Yip and Ng 2000], the authors proposed that to partly decrypt the configuration file, in order to increase the debugging effort during the reverse engineering. If an attacker copies the partly decrypted file, the non-decrypted functionality is available, whereas the one decrypted is not. Thus, the attacker tries to find the errors in the design not aware of the fact, that they are caused through the encrypted part of the configuration. Most likely an attacker with little resources, would have dropped the reverse engineering effort, when realizing that parts are decrypted (which he did not do because he did not know). However, this approach adds hardly any extra complexity to an attack if we assume that an attacker has a lot of resources. In [Kelem and Burnham 2000] an advanced scenario is introduced where the different parts of the configuration file are encrypted with different keys. The 60RS family from Actel was the first attempt to have a key stored in the FPGA in order to be able to encrypt the configuration file before transmitting it to the chip. The problem was that every FPGA had the same key on board. This implies that if an attacker has one key he can get the secret information from all FPGAs. In [Kean 2001], the author discusses some scenarios where depending on the manufacturing cost, more than one key is stored in the FPGA.

An approach in a completely different direction would be to power the whole SRAM FPGA with a battery, which would make transmission of the configuration file after a power loss unnecessary. This solution does not appear practical, however, because of the power consumption of FPGAs. Hence, a combination of encryption and battery power provides a possible solution. Xilinx addresses this with an on-chip 3DES decryption engine in its Virtex II [Xilinx Inc.] (see also [Pang et al. 2000]), where only the two keys are stored in the battery powered memory. Due to the fact that the battery powers only a very small memory cells, the battery is limited only by its own life span.

4.3 Preventing the Physical Attack

To prevent physical attacks, one has to make sure that the retention effects of the cells are as small as possible, so that an attacker can not detect the status of the cells. Already after storing a value in a SRAM memory cell for 100–500 seconds, the access time and operation voltage will change [van der Pol and Koomen 1990]. Furthermore, the recovery process is heavily dependant on the temperature: 1.5 hours at $75°C$, 3 days at $50°C$, 2 month at $20°C$, and 3 years at $0°C$ [Gutmann

2001]. The solution would be to invert the data stored periodically or to move the data around in memory. Cryptographic applications cause also long-term retention effects in SRAM memory cells by repeatedly feeding data through the same circuit. One example is specialized hardware that uses always the same circuits to feed the secret key to the arithmetic unit [Gutmann 2001]. Neutralization of this effect can be achieved by applying an opposite current [Tao et al. 1993] or by inserting dummy cycles into the circuit [Gutmann 2001]. In terms of FPGA application, it is very costly or even impractical to provide solutions like inverting the bits or changing the location for the whole configuration file. A possibility could be that this is done only for the crucial part of the design, like the secret keys. Counter techniques such as dummy cycles and opposite current approach can be carried forward to FPGA applications.

In terms of flash/EEPROM memory cell, one has to consider that the first write/erase cycles causes a larger shift in the cell threshold [San et al. 1995] and that this effect will become less noticeably after ten write/erase cycles [Haddad et al. 1989]. Thus, one should program the FPGA about 100 times with random data, to avoid these effect (suggested for flash/EEPROM memory cells in [Gutmann 2001]). The phenomenon of overerasing flash/EEPROM cells can be minimized by first programming all cells before deleting them.

4.4 Preventing the Readback Attack

The readback attack can be prevented with the security bits set, as provided by the manufactures, see Section 3.3. If one wants to make sure that an attacker is not able to apply fault injection, the FPGA has to be embedded into a secure environment, where after detection of an interference the whole configuration is deleted or the FPGA is destroyed.

4.5 Preventing the Side Channel Attack

In recent years, there has been a lot of work done to prevent side-channel attacks, see for example [Kocher et al. 1999; Chari et al. 1999; Chari et al. 1999; Goubin and Patarin 1999; Clavier et al. 2000; Clavier and Coron 2000; Shamir 2000]. The methods can generally be divide into software and hardware countermeasures, with the majority of proposals dealing with software countermeasures. “Software” countermeasures refer primarily to algorithmic changes, such as masking of secret keys with random values, which are also applicable to implementations in custom hardware or FPGA. Hardware countermeasures often deal either with some form of power trace smoothing or with transistor-level changes of the logic. Neither seem to be easily applicable to FPGAs without support from the manufacturers. However, some proposals such as duplicated architectures might work on todays FPGAs.

5. A WORD REGARDING METRICS

After considering the suitability of FPGAs for security applications from a systems perspective, we will try to give an overview of the state-of-the-art in cryptographic algorithm implementations on FPGAs. To this end, we first consider the metrics that we will use to measure performance in different implementations.

Traditionally, when evaluating the performance of cryptographic implementations emphasis has been made first on the throughput of the implementation and, second, on the amount of hardware resources consumed to achieve the previously mentioned throughput. As pointed out in [Elbirt et al. 2001], there is not a widely accepted metric to measure the hardware costs associated with a giving throughput in an FPGA implementation. This is evident when one looks at the many variables used in describing area results in many publications. Possibilities include: number of configurable logic blocks (CLBs), number of Look-Up Tables (LUTs) and flip-flops (FFs); number of equivalent logic gates; and number of CLB slices. It is important to point out that the number of equivalent logic gates does not provide an accurate measure of the extent to which hardware resources have been used in an FPGA. In particular, hardware resources within CLB slices may not be fully utilized by the place and route tools to relieve routing congestion, resulting in an increase in the number of CLB slices but not an equivalent increase in logic gates. Thus, following the conventions in [Elbirt et al. 2001; Gaj and Chodowiec 2000; 2001; Preneel et al. 2003], we have tried to use CLB slices whenever the numbers are available, and LUTs and FFs whenever CLB slices are not available.

In addition, since hardware costs are usually associated with area and throughput is considered one of the most important characteristic in a cryptographic implementation, we have also included the Throughput Per CLB slice (TPS) metric, used in [Elbirt et al. 2001; Preneel et al. 2003] and defined as:

$$\text{TPS} := \frac{\text{Encryption Rate}}{\#\ \text{CLB Slices Used}}$$

When appropriate, the TPS metric will be normalized and only used when the number of CLB slices and the throughput are explicitly stated in the work being cited. Therefore, the optimal implementation will achieve the highest throughput in the least amount of area and, thus, the highest TPS, behaving inversely to the more traditional time-area product used in VLSI implementations. At this point, we must caution the reader against using the TPS to compare implementations on different FPGA devices. Different FPGA devices, even within the same family, yield different timing results as a function of available logic and routing resources, both of which change based on the die size of a given FPGA. This is true to a greater extent if one attempted to compare different FPGA devices since now, even the architecture of the device is completely different. Nevertheless, we think the TPS is useful to compare implementations of different algorithms on the same device because this might shine some light into which algorithm is better suited for a given device. In addition, the TPS is also useful to compare different implementation methodologies of the same algorithm implemented on the same device because this helps in deciding which are the best architectures for a given algorithm based on the time-area cost of the implementation.

6. SYMMETRIC-KEY ALGORITHM IMPLEMENTATIONS ON FPGAS

We begin this section by describing DES implementations on FPGAs. Although single DES expired as a federal standard in 1998 and it can only be used in legacy systems, it continues to be the most widely deployed symmetric-key algorithm, including its variant triple-DES which coexists as a federal standard with the AES [American National Standards Institute 1998; U.S. Department of Commerce/National Institute of Standards and Technology 1999].

6.1 DES on FPGAs

The first published implementation of DES on an FPGA achieved a throughput of 24 Mbits/sec [Leonard and Magione-Smith 1997]. However, this implementation required generation of key-specific circuitry for the target FPGA, a Xilinx XC4013-14, requiring recompilation of the implementation for each key. Until recently, the best performance in an FPGA implementation of DES has achieved is a throughput of 402.7 Mbits/sec when operating in ECB mode using a Xilinx XC4028EX-3 FPGA [Kaps and Paar 1998; Kaps 1998; Kaps and Paar 1999]. This implementation took advantage of pipeline design techniques to maximize the system clock frequency at the cost of pipeline latency cycles. However, with the advent of run-time reconfiguration and more technologically advanced FPGAs, implementations with throughputs in the range of ASIC performance values has been achieved. The most recent DES implementation employing run-time reconfiguration achieved a throughput of 10.752 Gbits/sec when operating in ECB mode using a Xilinx Virtex XCV150-6 FPGA [Patterson 2000b]. The use of the Xilinx run-time reconfiguration software application JBits^{TM} allow for real time key-specific compilation of the bit-stream used to program the FPGA, resulting in a smaller and faster design (which operated at 168 MHz) as compared to the design in [Kaps and Paar 1998; Kaps 1998; Kaps and Paar 1999] (which operated at 25.19 MHz). However, the time required for run-time reconfiguration (RTR) each time that the key needs to be changed is in the order of tens of milliseconds. Most recently, there were two DES implementations which have achieved Gbit per second speeds for non-feedback modes of operation. The Free-IP project offers DES VHDL-implementations for feedback and non-feedback modes. They are capable of achieving 3 Gbit/sec encryption rates in non-feedback modes and 130 Mbit/sec in feedback modes. The second and by far the fastest FPGA implementation available in the literature (achieving a 12 Gbit/sec throughput when operating in ECB mode using a Xilinx Virtex-E XCV300E-8 FPGA) is the one described in [Trimberger et al. 2000]. This implementation did not make use of run-time reconfiguration –through careful optimization of the DES datapath and the use of a 48-stage pipeline, the implementation was able to achieve a clock frequency of 188.68 MHz, resulting in the improved performance versus the implementation in [Patterson 2000b]. Tables I and II summarize these results. Table I summarizes the performance of DES in feedback modes of operation, where pipelining can not be used. From these results, it is easy to see that one simple method to increase the throughput of an implementation is by using loop unrolling. In Table I, we have used the results in [Kaps 1998] rather than those in [Kaps and Paar 1998] because it allows us to compare the TPS metric when loop unrolling is used and when it is not[2]. Interestingly, the results seem to indicate that the price one pays for using loop unrollment increases at the same rate as the final throughput of the implementation.

Table II summarizes available DES implementations in non-feedback modes. In this case, one can easily use pipelining at the cost of additional area to improve the throughput of a given implementation. Notice that pipelining allows one to

[2] We would like to point out that the original metric used in [Kaps 1998; Kaps and Paar 1998] was CLBs. However, by looking at [Xilinx Inc. 1999] and [Xilinx Inc. 2001] one readily notices that one slice in a Virtex XCV400 chip is exactly the same as a CLB in a chip of the XC4000E family.

Table I. DES Feedback Implementation Performance on FPGAs

Reference	Chip	Area (slices)	Freq. (MHz)	Rate ($\frac{\text{Mbits}}{\text{sec}}$)	TPS ($\frac{\text{Mbps}}{\text{slices}}$)	Notes
Leonard and Magione-Smith [1997]	XC4025-4	938	4.9	20	0.02	Standard DES,
	XC4025-4	640	6.0	24	0.04	Key Specific
Kaps [1998]	XC4013E-3	262	23.9	91.2	0.35	Standard DES, 16 iterations
	XC4013E-3	443	18.5	141.3	0.32	Loop unrolled, 8 iterations
	XC4028EX-3	722	11.5	176	—	Loop unrolled, 4 iterations
[The Free-IP Project]	XCV400-6	731	32.5	130	—	

Table II. DES Non-Feedback Mode Implementation Performance on FPGAs

Reference	Chip	Area	Freq. (MHz)	Rate ($\frac{\text{Mbps}}{\text{slices}}$)	Notes
Kaps and Paar [1998]	XC4013E-3	433 slices	23.0	183.8	2-stage pipeline
	XC4028EX-3	741 slices	25.2	402.7	4-stage pipeline
[The Free-IP Project]	XCV400-6	2528 slices	47.7	3.1 Gbit/sec	
[Patterson 2000b]	XCV150-6	1584 slices	168	10.7	Requires Jbits and RTR
Trimberger et al.	XCV300-6	4216 LUTs, 1943 FFs,	10	6.4 Gbit/sec	16-stage pipeline
	XCV300-6	4216 LUTs, 5573 FFs	158.7	10.1 Gbit/sec	48-stage pipeline
	XCV300E-8	4216 LUTs, 1943 FFs	132.0	8.4 Gbit/sec	16-stage pipeline
	XCV300E-8	4216 LUTs, 5573 FFs	188.7	12.0 Gbit/sec	48-stage pipeline

obtain performance comparable to that attained by ASIC implementations (See for example [Wilcox et al. 1999] for a state-of-the-art DES ASIC implementation). In addition, we notice that except for [Patterson 2000b], all the other implementations implemented pipelined versions of DES and they did not require reconfiguration when changing keys. Nevertheless, the other implementations achieve comparable throughputs which allows us to conclude that it is not necessary to pay the price of a key-dependant implementation to achieve ASIC speeds on FPGAs. Having said that, one could envision key-specific FPGA implementations of DES [Leonard and Magione-Smith 1997; Patterson 2000b] to be useful in applications where area minimization is important (at the possible expense of throughput) and where large amounts of data are encrypted with the same key for extended periods of time.

6.2 AES Candidates on FPGAs

Multiple FPGA implementation studies have been presented for the AES candidate algorithm finalists [Dandalis et al. 2000a; 2000b; Elbirt et al. 2000; 2001; Gaj and

Chodowiec 2000; 2001; Weaver and Wawrzynek 2000], the results of which are found in Table III, for feedback modes of operation[3]. A similar exercise can be performed for non-feedback modes, but, we don't think it would add anything to this work. Note that [Elbirt et al. 2000] is a subset of the studied performed in [Elbirt et al. 2001] and that no throughput results are presented in [Weaver and Wawrzynek 2000]. The studies performed in [Elbirt et al. 2000; 2001; Gaj and Chodowiec 2000] used a Xilinx Virtex XCV1000-4 as the target FPGA. The study performed in [Dandalis et al. 2000a] used the Xilinx Virtex Family but did not specified which FPGA was used as the target device, this makes comparison with other implementations very hard, if not impossible.

Table III. AES Candidate Finalists FPGA Implementations in Feedback Mode of Operation.

Algorithm	Reference	Chip	Area	Freq.	Rate	TPS	Notes
			(slices)	(MHz)	($\frac{\text{Mbits}}{\text{sec}}$)	($\frac{\text{Mbps}}{\text{slices}}$)	
MARS	Dandalis et al. [2000a]	Xilinx Virtex	6896	—	101.9	—	incl. key schedule uses Block SelectRAM
	Gaj and Chodowiec [2001]	XCV1000-6	2744	—	61.0	0.02	iterative loop, no key schedule, incl. decryption & encryption
RC6	Dandalis et al. [2000a]	Xilinx Virtex	2650	—	112.9	—	incl. key schedule uses Block SelectRAM
	Gaj and Chodowiec [2001]	XCV1000-6	1137	—	142.7	0.13	iterative loop, no key schedule, incl. decryption & encryption
	Elbirt et al. [2001]	XCV1000-4	3189	19.8	126.5	0.04	speed opt., PP-2, no key-schedule no Block SelectRAM
	Gaj and Chodowiec [2000]	XC4085	1222 (CLBs)	7.2	43.1	—	iterative loop, no key schedule,
Rijndael	Dandalis et al. [2000a]	Xilinx Virtex	5673	—	353.0	—	incl. key schedule uses Block SelectRAM
	Gaj and Chodowiec [2001]	XCV1000-6	2507	—	414.2	0.17	iterative loop, no key schedule, incl. decryption & encryption
	Elbirt et al. [2001]	XCV1000-4	3528	25.3	294.2	0.08	speed opt., LU-1 no key schedule no Block SelectRAM
Serpent	Dandalis et al. [2000a]	Xilinx Virtex	2550	—	149.0	—	incl. key schedule uses Block SelectRAM
	Gaj and Chodowiec [2001]	XCV1000-6	4507	—	431.4	0.1	iterative loop, no key schedule, incl. decryption & encryption
	Elbirt et al. [2001]	XCV1000-4	7964	13.9	444.2	0.06	speed opt., LU-8 no key schedule no Block SelectRAM
Twofish	Dandalis et al. [2000a]	Xilinx Virtex	9363	—	173.1	—	incl. key schedule uses Block SelectRAM
	Gaj and Chodowiec [2001]	XCV1000-6	1076	—	177.3	0.17	iterative loop, no key schedule, incl. decryption & encryption
	Elbirt et al. [2001]	XCV1000-4	2695	16.0	127.7	0.05	area opt., LU-1 no key schedule no Block SelectRAM
	Gaj and Chodowiec [2000]	XC4085	907 (CLBs)	11.4	89.2	—	iterative loop, no key schedule,

In Table III, PP-2, LU-1, and LU-8, correspond to partial pipelining with two stages, one round loop unrolling, and eight rounds loop unrolling architectures, respectively. In addition, notice that the implementation of a one stage partial pipeline, an iterative looping architecture, a one round loop unrolled architecture are all equivalent [Elbirt et al. 2001]. Also, we have only calculated the TPS ratio for

[3] In Table III, we have not always included the implementation corresponding to highest throughput, but rather the one with the highest TPS ratio.

implementations which used the Xilinx Virtex XCV1000 as their hardware platform because it is the most common platform among published works, thus making a comparison somewhat reasonable. One can see that most implementations achieve similar throughputs for the same algorithms. In addition, if one were to choose algorithms based on their throughput all authors would agree that Serpent would win followed closely by Rijndael, Twofish and RC6, and MARS at the end[4]. The most interesting effect is that [Gaj and Chodowiec 2001] achieve similar performance as other implementations at the cost of half the area, which is due, according to the authors, to resource sharing. If we were to perform a similar exercise in terms of the TPS ratio, then Rijndael would win followed by Serpent and Twofish, and finally, RC6 and MARS. One reason for RC6 and MARS to have the poorest performance when implemented on FPGAs is their use of a multiplier in their round function. We refer to [Gaj and Chodowiec 2001; Elbirt et al. 2001; Nechvatal et al. 2000] for more in depth architecture and performance comparisons of the AES candidates.

FPGA implementations of individual candidate algorithms (both finalists and non-finalists) have also been performed. Implementations of CAST-256 achieved throughputs of 11.03 Mbits/sec using a Xilinx Virtex XCV1000-4 [Elbirt 1999] and 13 Mbits/sec using a Xilinx XC4020XV-9 [Riaz and Heys 1999]. An RC6 implementation achieved a throughput of 37 Mbits/sec using a Xilinx XC402OXV-9 [Riaz and Heys 1999]. A Serpent implementation using a Xilinx Virtex XCV1000-4 achieved a throughput of 4.86 Gbits/sec [Elbirt and Paar 2000]. When targeted to a Xilinx Virtex-E XCV400E-8, a Serpent implementation achieved a throughput of 17.55 Gbits/sec through the use of the Xilinx run-time reconfiguration software application JBits^{TM} which allowed for real time key-specific compilation of the bit-stream used to program the FPGA [Patterson 2000a]. This run-time reconfiguration resulted in a smaller and faster design (which operated at 137.15 MHz) as compared to the design in [Elbirt and Paar 2000] (which operated at 37.97 MHz). When implemented using an FPGA from the Altera Flex 10KA Family, the Serpent algorithm achieved a maximum throughput of 301 Mbits/sec [Bora and Czajka 1999], however, it is important to note that the implementation in [Bora and Czajka 1999] implements 8 of the Serpent's algorithm thirty two rounds while the implementations in [Elbirt and Paar 2000] and [Patterson 2000a] implement all the rounds of the Serpent algorithm. Note that all of the presented throughput values are for non-feedback modes of operation.

Multiple implementations of Rijndael, the AES, have been presented using both Xilinx and Altera FPGAs [Fischer and Drutarovsky 2001; McLoone and McCanny 2001]. The implementation in [McLoone and McCanny 2001] achieves a throughput of 6.956 Gbits/sec using a Xilinx Virtex-E XCV3200E-8. Utilizing ROM to implement the Rijndael Byte-Sub operation resulted in a significant increase in throughput and decrease in area as compared to implementations in [Dandalis et al. 2000a; 2000b; Elbirt et al. 2000; Elbirt and Paar 2001; Gaj and Chodowiec 2000], at the expense of BRAM Blocks which the previous implementations did not use. When targeting the more advanced Altera APEX20KE200-1, Rijndael implementations achieved throughputs ranging from 570 Mbits/sec to 964 Mbits/sec depending on

[4] [Dandalis et al. 2000a] and [Gaj and Chodowiec 2001; Elbirt et al. 2001] would interchange the placing of Rijndael and Serpent.

the implementation methodology [Fischer and Drutarovsky 2001]. Four recent implementations are also worth mentioning. The implementations described in [Standaert et al. 2003a; 2003b] and [Järvinen et al. 2003] achieve throughput rates of 11.8 and 17.8 Gbits/s, respectively. These throughputs are only achieved through pipelining and thus are not suitable for feedback modes of operation. Notice that [Järvinen et al. 2003] achieve such high throughputs for a 128-bit key Rijndael implementation. The work presented in [Standaert et al. 2003a; 2003b] is also interesting because it compares different implementation options (Look-up table based implementations, RAM-based implementations, and composite field implementations) and proposed some heuristics to evaluate the hardware efficiency at different steps of the design process which result on particularly efficient implementations. Finally, we include the work presented in [Chodowiec and Gaj 2003] as its target is the implementation of a resource efficient AES core on FPGAs. Such work has not been considered extensively in the literature (usually designs are optimized for speed rather than area when targeting FPGAs). In addition, they proposed a new way of implementing the MixColumns and InvMixColumns transformations which reduces area and might be interesting in its own right. The work in [Chodowiec and Gaj 2003] achieves data streams of 150 Mbits/sec for encryption and decryption on a low-cost Xilinx Spartan II FPGA using 222 slices and 3 BRAMs.

7. ASYMMETRIC-KEY ALGORITHM IMPLEMENTATIONS ON FPGAS

Most public–key schemes are based on modular exponentiation (RSA [Rivest et al. 1978] and Discrete Logarithm (DL) based systems [Diffie and Hellman 1976; U.S. Department of Commerce/National Institute of Standard and Technology 2000b]) or point multiplication (Elliptic Curve Cryptosystems [Miller 1986b; Koblitz 1987; Menezes and Johnson 1999; U.S. Department of Commerce/National Institute of Standard and Technology 2000b]). Both operations are in their most basic forms performed via the binary method for exponentiation or one of its variants [Gordon 1998]. The atomic operation in the binary method for exponentiation is either modular multiplication, in the case of RSA and DL-based systems, or point addition, in the case of ECC, which in turn is performed through a combination of multiplications and additions on the field of definition of the elliptic curve. Thus, the first part of this section is mainly concerned with how to perform modular multiplication efficiently on FPGAs. The second part of this section treats the case of elliptic curves.

7.1 Notation

We will refer to long numbers with capital letters and to their digits in radix-b representation with lower-case letters. So for example, we would write an n-digit number in base b as $A = \sum_{i=0}^{n-1} a_i b^i$ with $b \geq 2$ and $0 \leq a_i < b$. Notice that unless otherwise stated we always assume unsigned operands.

7.2 Modular Multiplication

The problem of modular multiplication and, more specifically, the problem of modular reduction has been extensively studied since it is a fundamental building block of any cryptosystem. Among the algorithms that have been proposed we find:

—*Sedlak's Modular Reduction.* Originally introduced in Sedlak [1987], this algorithm is used by Siemens, in the SLE44C200 and SLE44CR80S microprocessors, to perform modular reduction [Naccache and M'Raïhi 1996]. Sedlak notices that the algorithm improves the reduction complexity by an average factor of 1/3 when compared to the basic bit-by-bit reduction.

—*Barret's Modular Reduction.* It was originally introduced in [Barrett 1986], in the context of implementing RSA on a DSP processor. Suppose that you want to compute $X \equiv R \bmod M$ for some modulus M. Then, we can re-write X as $X = Q \cdot M + R$ with $0 \leq R < M$, which is a well known identity from the division algorithm [Menezes et al. 1997, Definition 2.82]. Thus

$$R = X \bmod M = X - Q \cdot M \tag{1}$$

Barret's basic idea is that one can write Q in (1) as:

$$Q = \lfloor X/M \rfloor = \left\lfloor \left(X/b^{n-1}\right) \left(b^{2n}/M\right) \left(1/b^{n+1}\right) \right\rfloor . \tag{2}$$

In particular, Q can be approximated by

$$\widehat{Q} = Q_3 = \left\lfloor \left\lfloor \left(X/b^{n-1}\right) \right\rfloor \left(b^{2n}/M\right) \left(1/b^{n+1}\right) \right\rfloor$$

Notice that the quantity $\mu = b^{2n}/M$ can be precomputed when performing many modular reductions with the same modulus, as is the case in cryptographic algorithms. Having precomputed μ, the expensive computations in the algorithm are only divisions by powers of b, which are simply performed by right-shifts, and modular reduction modulo b^i, which is equivalent to truncation. We refer to [Menezes et al. 1997, Section 14.3.3] for further discussion of implementation issues regarding Barret reduction, and to [Dhem 1994; 1998] for improvements over the original algorithm.

—*Brickell's Modular Reduction.* Originally introduced in [Brickell 1982], is dependent on the utilization of carry-delayed adders [Norris and Simmons 1981] and combines a sign estimation technique (See for example [Koç and Hung 1991]) and Omura's modular reduction [Omura 1990].

—*Quisquater's Modular Reduction.* Quisquater's algorithm, originally presented at [Quisquater ; 1992], can be thought of as an improved version of Barret's reduction algorithm. [Benaloh and Dai ; Walter 1991] have proposed similar methods. In addition, the method is used in the Phillips smart-card chips P83C852 and P83C855, which use the CORSAIR crypto-coprocessor [D. De Waleffe and J.-J. Quisquater 1990; Naccache and M'Raïhi 1996] and the P83C858 chip, which uses the FAME crypto-coprocessor [Ferreira et al. 1996]. Quisquater's algorithm, as presented in [D. De Waleffe and J.-J. Quisquater 1990], is a combination of the interleaved multiplication reduction method (basically, combine a normal multiprecision algorithm with modular reduction, making use of the distributivity property of the modular operation) and a method that makes easier and more accurate the estimation of the quotient Q in (1).

—*Montgomery Modular Multiplication.* The Montgomery algorithm, originally introduced in [Montgomery 1985], is a technique that allows efficient implementation of the modular multiplication without explicitly carrying out the modular reduction step. We discuss it in detail as it is the most widely used algorithm for modular multiplication in the literature.

7.3 Montgomery Modular Multiplication

The idea behind Montgomery's algorithm is to transform the integers in M-residues and compute the multiplication with these M-residues. At the end, one transforms back to the normal representation. As with Quisquater's and Barret's method, this approach is only beneficial if we compute a series of multiplications in the transform domain (e.g., modular exponentiation). The Montgomery reduction algorithm is as follows: Given integers M and R with $R > M$ and $\gcd(M,R) = 1$, $M' \equiv -M^{-1} \bmod R$. Let T be an integer such that $0 \leq T < MR$. If $Q \equiv TM' \bmod R$, then $Z = (T + QM)/R$ is an integer and furthermore, $Z \equiv TR^{-1} \bmod M$. Notice that our description is just the reduction step involved in a modular multiplication. The multiplication step can be carried out via multi-precision multiplication (see for example [Menezes et al. 1997, Chapter 14]). As with previous algorithms, one can interleave multiplication and reduction steps. The result is shown in Algorithm 1. In practice R is a multiple of the word size of the processor and a power of two. This means that M, the modulus, has to be odd (because of the restriction $\gcd(M,R) = 1$) but this does not represent a problem as M is a prime or the product of two primes (RSA) in most practical cryptographic applications. In addition, choosing R a power of 2, simplifies the computation of Q and Z as they become simply truncation (modular reduction by R) and right shifting (division by R). Notice that $M' \equiv -M \bmod R$. In [Dussé and Kaliski 1990] it is shown that if $M = \sum_{i=0}^{n-1} m_i b^i$, for some radix b typically a power of two, and $R = b^n$, then M' can be substituted by $m'_0 = -M^{-1} \bmod b$. In [Eldridge and Walter 1993], the authors simplify the combinatorial logic needed to implement Montgomery reduction.

Algorithm 1 Montgomery Multiplication Algorithm

```
Require: $X = \sum_{i=0}^{n-1} x_i b^i$, $Y = \sum_{i=0}^{n-1} y_i b^i$, $M = \sum_{i=0}^{n-1} m_i b^i$, with $0 \leq X, Y < M$,
    $b > 1$, $m' = -m_0^{-1} \bmod b$, $R = b^n$, $\gcd(b, M) = 1$
Ensure: $Z = X \cdot Y \cdot R^{-1} \bmod M$
$Z \leftarrow 0$ {where $Z = \sum_{i=0}^{n} z_i b^i$}
for $i = 0$ to $n - 1$ do
    $Z \leftarrow (Z + x_i \cdot Y + q_i \cdot M)/b$
    $q_i \leftarrow (z_0 + x_i \cdot y_0)\, m' \bmod b$
end for
if $Z \geq M$ then
    $Z \leftarrow Z - M$
end if
Return($Z$)
```

The idea in [Eldridge and Walter 1993], is to shift Y by two digits (i.e., multiply Y by b^2) and thus, make q_i in Step 4 of Algorithm 1 independent of Y. Notice that one could have multiplied Y by b instead of b^2 and have also obtained a q_i independent of Y. However, by multiplying Y by b^2, one gets q_i to be dependent only on the partial product Z and on the lowest two digits of the multiple of M (i.e. $q_i \cdot M$). The price of such a modification is two extra iterations of the for-loop for which the digits of X are zero. The architecture proposed by [Eldridge and

Walter 1993] is only considered for the case $b = 2$ and estimated to be twice as fast as previous modular multiplication architectures at the time of publication.

7.4 Higher Radix Montgomery Modular Multiplication

In [Vuillemin et al. 1996; Shand and Vuillemin 1993] modular exponentiation architectures are implemented on an array of 16 Xilinx 3090 FPGAs. Their design uses several speed-up methods [Shand and Vuillemin 1993] including the Chinese remainder theorem, asynchronous carry completion adder, and a windowing exponentiation method. Some of the improvements are:

—Avoid having to perform a subtraction after every modular product of the exponentiation algorithm by letting all intermediate results have *two extra bits* of precision.[Shand and Vuillemin 1993] also show that even allowing for the two extra bits of precision, one can always manage to work with intermediate results no larger than n- digits if $M < b^n/4$ and $X, Y \leq 2M$.

—A second improvement is the use of a radix $b = 2^2$, which permits for a trivial computation of the quotient q_i in Step 4 of Algorithm 1 and it allows for the use of Booth recoded multiplications (this doubles the multipliers performance compared to $b = 2$ at an approximate 1.5 increase in hardware complexity). Higher radices, which would offer better performance, were dismissed since they involve too great of a hardware cost and the computation of the quotient digits is no longer trivial.

—They re-write Montgomery's Algorithm in a similar way to [Eldridge and Walter 1993], to allow for pipeline execution, basically getting rid off of the q_i dependency on the least significant digit of the partial product Z. The cost for d levels of pipelining is d extra bits of precision and d more cycles in the computation of the final product.

The result of all these speedup methods, is an RSA secret decryption rate of over 600 Kbits/sec for a 512-bit modulus and of 165 Kbits/sec for a 1024-bit modulus, using the CRT. While the previous results make full use of the reconfigurability of the FPGAs (reconfigurability is required to recombine the result of the CRT computations), they derive a single gate-array specification whose size is estimated under 100K gates and speed over 1Mbit/sec for RSA 512-bit keys.

The main obstacle to the use of higher radices in the Montgomery algorithm is that of the quotient determination. In [Orup 1995], the author presents a method which avoids quotient determination all together and thus makes higher-radix Montgomery practical. The price to pay for avoiding quotient determination is more precision and at most one more iteration in the main loop of the algorithm. The final improvement in [Orup 1995] is the use of quotient pipelining. Unlike [Shand and Vuillemin 1993], is able to achieve quotient pipelining only at the cost of extra loop iterations and no extra precision.

As an example, Orup [1995] considers an architecture with 3 pipeline stages and a radix $b = 2^8$. The author estimate the critical path of the architecture to be no more than 5ns assuming 1995 CMOS technology. It is also assumed the use of a redundant representation for the intermediate values of the Montgomery multiplier. However, the outputs have to be converted back to non-redundant representation using a carry-ripple adder with an asynchronous carry completions detection circuit as proposed in [Shand and Vuillemin 1993]. With these techniques, the author estimates the time of one 512-bit modular multiplication at 415 nsec. Using the left-to-right binary method for exponentiation, one 512-bit exponentiation would take 319 μsec which corresponds to a 1.6 Mbit/sec throughput. If instead, one uses the right-to-left binary exponentiation algorithm, one can perform multiplications and squarings in parallel as shown in [Orup and Kornerup 1991], thus achieving a factor of two speedup, i.e., more than 2.4 Mbit/sec throughput. This is four times faster than the implementation of [Shand and Vuillemin 1993] which at the time was the fastest. Furthermore, if the modulus is composite, as in the RSA case, and its prime factorization is known, it is possible to obtain a factor of four speedup through the use of the CRT as in [Shand and Vuillemin 1993].

In [Blum and Paar 1999], the authors implemented a version of Montgomery's algorithm optimized for a radix two hardware implementation. Blum and Paar [2001] extends [Orup 1995] to reconfigurable hardware, a systolic array architecture as presented in [Kornerup 1994], and following [Orup 1995] high radix hardware implementations of modular exponentiation. There had been a number of proposals for systolic array architectures for modular arithmetic but, to our knowledge, [Blum and Paar 1999; 2001] were the first implementations that have been reported. For the exact design and technical details we refer the reader to [Blum and Paar 1999; Blum 1999; Blum and Paar 2001]. Here, however, summarize their results. As target devices, [Blum and Paar 1999; 2001] used the Xilinx XC40250XV, speedgrade -09, 8464 CLBs, for the larger designs (> 5000 CLBs), and the XC40150XV, speedgrade -08, 5184 CLBs, for the smaller designs. Table IV shows our results for a full length modular exponentiation, i.e., an exponentiation where base, exponent, and modulus have all the same bit length. We notice that [Blum and Paar 1999; 2001] both use the right-to-left method for exponentiation. Table V shows [Blum and Paar 2001] RSA encryption results. The encryption time is calculated for the Fermat prime $F_4 = 2^{16} + 1$ exponent [Knuth 1981], requiring $2 \cdot 19(n + 2)$ clock cycles for the radix 2 design [Blum and Paar 1999], and $2 \cdot 19(n + 8)$ clock cycles if the radix 16 design is used, where the modulus has $n - 2$ bits.

Table IV. CLB usage and execution time for a full modular exponentiation

	512 bit		768 bit		1024 bit	
Radix	C (CLBs)	T (msec)	C (CLBs)	T (msec)	C (CLBs)	T (msec)
2 [Blum and Paar 1999]	2555	9.38	3745	22.71	4865	40.05
16 [Blum and Paar 2001]	3413	2.93	5071	6.25	6633	11.95

Table V. Application to RSA: Encryption

	512 bit		1024 bit	
Radix	C (CLBs)	T (msec)	C (CLBs)	T (msec)
2 [Blum and Paar 1999]	2555	0.35	4865	0.75
16 [Blum and Paar 2001]	3413	0.11	6633	0.22

For decryption, [Blum and Paar 2001] apply the Chinese remainder theorem [Quisquater and Couvreur 1982]. They either decrypt m bits with an $m/2$ bit architecture serially, or with two $m/2$ bit architectures in parallel. The first approach uses only half as many resources, the latter is almost twice as fast. A little time is lost here because of the slower delay specifications of the larger devices.

Table VI. Application to RSA: Decryption

	512 bit 2 × 256 serial		512 bit 2 × 256 parallel		1024 bit 2 × 512 serial		1024 bit 2 × 512 parallel	
Radix	C (CLBs)	T (msec)	C (CLBs)	T (msec)	C (CLBs)	T (msec)	C (CLBs)	T (msec)
2 [Blum and Paar 1999]	1307	4.69	2614	2.37	2555	18.78	5110	10.18
16 [Blum and Paar 2001]	1818	1.62	3636	0.79	3413	5.87	6826	3.10

7.5 Elliptic Curves Cryptosystems over $GF(p)$

In this section, we first provides a brief introduction to elliptic curve point addition and doubling. Additional information can be found in [Miller 1986b; Koblitz 1987; Blake et al. 1999]. An elliptic curve E over $GF(p)$ is the set of solutions $P = (x, y)$ which satisfy the Weierstrass equation:

$$E: \; y^2 = x^3 + Ax + B \pmod{p} \tag{3}$$

where $A, B \in GF(p)$ and $4A^3 + 27B^2 \not\equiv 0 \bmod M$ with $M > 3$, together with the point at infinity $\mathcal{O}$. It is well known that the points on elliptic curve form a group under an addition operation which is defined as follows. Let $P = (x_0, y_0) \in E$; then $-P = (x_0, -y_0)$. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$. If $Q = (x_1, y_1) \in E$ and $Q \neq -P$, then $P + Q = (x_2, y_2)$, where

$$x_2 = \lambda^2 - x_0 - x_1 \tag{4}$$

$$y_2 = \lambda(x_1 - x_2) - y_1 \tag{5}$$

and

$$\lambda = \begin{cases} \frac{y_0 - y_1}{x_0 - x_1} & \text{if } P \neq Q \\ \\ \frac{3x_1^2 + A}{2y_1} & \text{if } P = Q \end{cases} \tag{6}$$

This coordinate representation is known as affine representation. However, in many applications it is more convenient to represent the points P and Q in projective coordinates. This is advantageous when inversion is very computationally expensive compared to multiplication in the finite field $GF(p)$. Thus, algorithms for projective coordinates trade inversions in the point addition and in the point double operations for a larger number of multiplications and a single inversion at the end of the algorithm. This inversion can be computed via exponentiation using the fact that $A^{-1} \bmod M \equiv A^{M-2} \bmod M$, for prime modulus M. In projective coordinates, a point $P = (x, y)$ is represented as $P = (X, Y, Z)$ where $X = x$, $Y = y$, and $Z = 1$.

To convert from projective coordinates back to the affine ones, we use the following relations:

$$x = \frac{X}{Z^2}, \ \ y = \frac{Y}{Z^3}$$

Finally, one can obtain expressions equivalent to (6)for doubling and addition operations in projective coordinates. We refer to [Chudnovsky and Chudnovsky 1986; P1363 2000] for the actual algorithms. One can achieve a point doubling in the general case with 10 finite field multiplications [5] and in as little as 3 multiplications for special parameters[Chudnovsky and Chudnovsky 1986]. Similarly, addition requires 16 field multiplications in the general case and only 11 when one of the points being added is constant, i.e., the case of cryptographic applications [P1363 2000].

7.6 FPGA Processor Architecture for ECC over $GF(p)$

Orlando and Paar [2001] proposes a new elliptic curve processor (ECP) architecture for the computation of point multiplication for curves defined over fields $GF(p)$. Point multiplication is defined as the product kP, where k is an integer, P is a point on the elliptic curve, and by multiplication we mean that P is added to itself k times. We emphasize that there is no multiplication operation on the elliptic curve, only additions and doublings of points $P \in E$. The ECP is best suited for the computation of point multiplications using projective coordinates. Inversions are computed via Fermat's Little Theorem and multiplication via Montgomery reduction.

The ECP has a scalable architecture in terms of area and speed specially suited for memory-rich hardware platforms such a field programmable gate arrays (FPGAs). This processor uses a new type of high-radix Montgomery multiplier that relies on the precomputation of frequently used values and on the use of multiple processing engines. The ECP consists of three main components. These components are the main controller (MC), the arithmetic unit controller (AUC), and the arithmetic unit (AU). The MC is the ECP's main controller. It orchestrates the computation of kP and interacts with the host system. The AUC controls the AU. It orchestrates the computation of point additions/subtractions, point doubles, and coordinate conversions. It also guides the AU in the computation of field inversions. Both the MC and the AUC execute their respective operations concurrently and they have the capability of executing one instruction per clock cycle. The AU incorporates a multiplier, an adder (or adders), and a register file, all of which can operate in parallel on different data. The AU's large register set supports algorithms that rely on precomputations.

As with systems based on RSA or the DL problem in finite fields, in ECC-based systems, multiplication is also the most critical operation in the computation of elliptic curves point multiplications. The elliptic curve processor (ECP) introduced in [Orlando and Paar 2001] develops a new multiplier architecture that draws from [Orup 1995; Frecking and Parhi 1999] an approach for high radix multiplication, from [Shand and Vuillemin 1993; Orup 1995] the ability to delay quotient resolution,

[5] Note that in this context, the complexity of adding or doubling a point on an elliptic curve is usually given by the number of field multiplications and inversions (if affine coordinates are being used), field additions are relatively cheap operations compared to multiplications or inversions.

and from [Blum 1999] the use of precomputation. In particular, this work extends the concept of precomputation. The resulting multiplier architecture is a high-radix, precomputation-based modular multiplier, which supports positive and negative operands, Booth recoding and pre-computation.

Orlando and Paar [2001] developed a prototype that implemented the double-and-add algorithm using the projective coordinates algorithms defined in [P1363 2000] for point addition and point double operations on a Xilinx's XCV1000E-8-BG680 (Virtex E) FPGA. This prototype was programmed to support the field $GF(2^{192} - 2^{64} - 1)$, which is one of the fields specified in [U.S. Department of Commerce/National Institute of Standard and Technology 2000b]. To verify the ECP's architectural scalability to larger fields, a modular multiplier for fields as large as $GF(2^{521} - 1)$ was also prototyped. The ECP prototype for $GF(2^{192} - 2^{64} - 1)$ used 11,416 LUTs, 5,735 Flip-Flops, and 35 BlockRAMS. The frequency of operation of the prototype was 40 MHz for 192 bit operands and 37.3 MHz for the 521-bit multiplier. The authors in [Orlando and Paar 2001] point out that assuming that the ECP is coded in a form that extracts 100% throughput from its multiplier, it will compute a point multiplication for an arbitrary point on a curve defined over $GF(2^{192} - 2^{64} - 1)$ in approximately 3 msec. This estimate ignores the processing cost of additions and overhead operations and it assumes the computation of $17m$ multiplications per point multiplication: $15.5m$ for the point double and the point add operations and $1.5m$ for the inverse required in the conversion to affine coordinates.

8. OPEN PROBLEMS

At this point we would like to provide a list of open questions and problems regarding the security of FPGAs. If answered, such solutions would allow stand-along FPGAs with much higher security assurance than currently available.

Physical attacks. There has not been any published physical attacks against FPGAs. In the following we list some open questions regarding the physical security of FPGAs, that should be answered from industry and research community:

—Have FPGAs really more resistant against physical attacks than ASICs?

—What kind of technical equipment is necessary to accomplish a physical attack against FPGAs?

—How much effort is needed to successfully attack an FPGA?

—What are the cost of such an attack?

—How much time and man-years are needed to perform such an attack?

—Does the internal structure of the FPGA (i.e. logic elements, connections and so on) make the life of the attacker more difficult? or does it even be helpful for an attack?

Again, we think it would be very interesting to study this problems.

Side channel attacks. Side channel attacks on FPGAs should be investigated with the same intensity as it has been done with processor and ASIC platforms. We assume that many characteristics from ASIC attacks carry over, but some features will certainly be different. One extremely interesting question in this context is whether there are design strategies that make FPGA designs less vulnerable against power analysis attacks. If so, can we integrate such strategies in the design tools?

Fault injection. There appears to be no published attempt to perform this kind of attack against FPGAs. It seems very likely that one can use, for example, radiation or glitches to alter the contents of FPGAs. However, for attacks that need to target specific components of a design, the small features of current FPGAs together with the uncertainty of the location of the design might make those fault injection attacks a more formidable task in the FPGA case. We think that this subject is worth of further investigation.

Key management for configuration encryption. On-chip decryption of an encrypted configuration file would have benefits well beyond cryptographic applications. One major problem to overcome is the key management. The specific problems include who assigns the keys, who keeps track of them, secure storage of the key, and possibly zeroization of the key when an attack is detected.

battery:

Secure deletion. The configuration time of an FPGA transcends more than one minute and hence it is not possible to overwrite or delete the configuration after an attack was detected. This is bad, especially if the configuration of the FPGA includes any proprietary algorithms which should be kept secret. There should be research on how to securely delete the design information. A related problem is on-chip tamper detection for FPGAs.

9. CONCLUSIONS

This contribution analyzed possible attacks against the use of FPGA in security applications. Black box attacks do not seem to be feasible for state-of-the-art FPGAs. However, it seems very likely for an attacker to get the secret information stored in a FPGA, when combining readback and fault injection attacks. Cloning of SRAM FPGA and reverse engineering depend on the specifics of the system under attack, and they will probably involve a lot of effort, but this does not seem entirely impossible. Physical attacks against FPGAs are very complex due to the physical properties of the semiconductors in the case of flash/SRAM/EEPROM FPGAs and the small size of AF cells. It appears that such attacks are even harder than analogous attacks against ASICs. Even though FPGA have different internal structures than ASICs with the same functionality, we believe that side-channel attacks against FPGAs, in particular power-analysis attacks, will be feasible too.

The second part of the contribution summarized the state-of-the-art in implementations of symmetric-key and public-key algorithms. Symmetric-key ciphers and, specially, DES and AES have been throughly investigated and different approaches to implementations exposed. The different approaches to modular multiplication, the main operation in most public-key cryptosystems, were briefly summarized and the Montgomery multiplication algorithm, by far the most popular reduction method, throughly studied. In addition, we describe the current tendencies in elliptic curve processor design, a growing are of interest, specially in the context of embedded systems.

It seems from our previous remarks that, while, the art of cryptographic algorithm implementation is reaching maturity, FPGAs as a security platform are not, and in fact, that they might be currently out of question for security applications. We don't think that is the right conclusion, however. It should be noted that many commercial ASICs with cryptographic functionality are also vulnerable to attacks similar to the ones discussed here. A commonly taken approach to prevent these attacks is to put the ASIC in a secure environment. A secure environment could, for instance, be a box with tamper sensors which triggers what is called "zeroization" of cryptographic keys, when an attack is being detected. Similar approaches are certainly possible for FPGAs too. (Another solution often taken by industry is not to care and to build cryptographic products with poor physical security, but we are not inclined to recommend this.)

REFERENCES

Actel Corporation. 2002. Design Security in Nonvolatile Flash and Antifuse. Avaialble at `http://www.actel.com/appnotes/DesignSecurity.pdf`.

Agrawal, D., Archambeault, B., Rao, J. R., and Rohatgi, P. 2002. The EM Side – Channel(s). In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 2523. Springer-Verlag, 29–45.

Ajluni, C. 1995. Two New Imaging Techniques to Improve IC Defect Indentification. *Electronic Design 43,* 14 (July), 37–38.

Algotronix Ltd. Method and Apparatus for Secure Configuration of a Field Programmable Gate Array. PCT Patent Application PCT/GB00/04988.

Altera Corporation 2000. *Nios Soft Core Embeded Processor*. Altera Corporation. Available at `http://www.altera.com/products/devices/nios/nio-index.html`.

Altera Corporation 2002a. *Excalibur Device Overview*. Altera Corporation. Available at `http://www.altera.com/products/devices/arm/arm-index.html`.

Altera Corporation 2002b. *Stratix FPGA Family*. Altera Corporation. Available at `http://www.altera.com/products/devices/dev-index.jsp`.

American National Standards Institute 1998. *ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation*. American National Standards Institute. Available at `http://webstore.ansi.org/ansidocstore/dept.asp?dept_id=80`.

Amphion. High Performance AES Encryption Cores. Available at `http://www.chipcenter.com/networking/images/prod/prod226.pdf`.

Anderson, R. and Kuhn, M. 1997. Low Cost Attacks on Tamper Resistant Devices. In *5th International Workshop on Security Protocols*, B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, Eds. Vol. LNCS 1361. Springer-Verlag, 125–136.

ANSI. 1981. *American National Standards Data Encryption Algorithm X3.92-1981.* American National Standards Association.

Aplan, J. M., Eaton, D. D., and Chan, A. K. 1999. Security Antifuse that Prevents Readout of some but not other Information from a Programmed Field Programmable Gate Array. United States Patent, Patent Number 5898776.

Aritome, S., Shirota, R., Hemink, G., Endoh, T., and Masuoka, F. 1993. Reliability Issues of Flash Memory Cells. *Proceedings of the IEEE 81,* 5 (May), 776–788.

Athanas, P. and Abbott, A. 1995. Real-Time Image Processing on a Custom Computing Platform. *IEEE Computer 28,* 2 (February), 16–24.

Austin, K. 1995. Data Security Arrangements for Semicondutor Programmable Devices. United States Patent, Patent Number 5388157.

Barrett, P. 1986. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology – CRYPTO '86*, A. M. Odlyzko, Ed. Vol. LNCS 263. Springer-Verlag, Berlin, Germany, 311–323.

Benaloh, J. and Dai, W. Fast modular reduction. Rump session of CRYPTO '95.

Biham, E. and Shamir, A. 1997. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology - CRYPTO '97*, B. Kaliski, Jr., Ed. Vol. LNCS 1294. Springer-Verlag, 513–525.

Blake, I., Seroussi, G., and Smart, N. 1999. *Elliptic Curves in Cryptography.* Cambridge University Press, London Mathematical Society Lecture Notes Series 265.

Blum, T. 1999. Modular exponentiation on reconfigurable hardware. M.S. thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA.

Blum, T. and Paar, C. 1999. Montgomery modular multiplication on reconfigurable hardware. In *Proceedigns of the 14th IEEE Symposium on Computer Arithmetic (ARITH-14)*. 70–77.

Blum, T. and Paar, C. 2001. High radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers 50,* 7 (July), 759–764.

Bondalapati, K. and Prasanna, V. 2002. Reconfigurable Computing Systems. *Proceedings of the IEEE*.

Boneh, D., DeMillo, R. A., and Lipton, R. J. 1997. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *Advances in Cryptology - EUROCRYPT '97*, W. Fumy, Ed. Vol. LNCS 1233. Springer-Verlag, 37–51.

Bora, P. and Czajka, T. 1999. Implementation of the Serpent Algorithm Using Altera FPGA Devices. Available at `http://csrc.nist.gov/encryption/aes/round2/pubcmnts.htm`.

Borriello, G. and Want, R. 2000. Embedded computation meets the world wide web. *Communications of the ACM 43,* 5 (May), 59–66.

Brickell, E. F. 1982. A fast modular multiplication algorithm with applications to two key cryptography. In *Advances in Cryptology — CRYPTO '82*, D. Chaum and R. L. Rivest and A. T. Sherman, Ed. Plenum Publishing, New York, USA, 51–60.

Buell, D., Arnold, J., and Kleinfelder, W. 1996. *Splash 2: FPGAs in a Custom Computing Machine.* John Wiley and Sons.

Chameleon Systems Inc. Available at `http://www.chameleonsystems.com/`.

Chari, S., Jutla, C. S., Rao, J. R., , and Rohatgi, P. 1999. A Cauttionary Note Regarding the Evaluation of AES Condidates on Smart Cards. In *Proceedings of the Second AES Candidate Conference (AES2).* Rome, Italy.

Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology — CRYPTO '99*, M. Wiener, Ed. Vol. LNCS 1666. Springer-Verlag, 398–412.

Chodowiec, P. and Gaj, K. 2003. Very Compact FPGA Implementation of the AES Algorithm. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, C. Walter, Ç. K. Koç, and C. Paar, Eds. Vol. LNCS 2779. Springer-Verlag, 319–333.

Chudnovsky, D. and Chudnovsky, G. 1986. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics 7,* 4, 385–434.

Clavier, C., Coron, J., and Dabbous, N. 2000. Differential Power Analysis in the Presence of Hardware Countermeasures. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, 252–263.

Clavier, C. and Coron, J.-S. 2000. On Boolean and Arithmetic Masking against Differential Power Analysis. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, 231 – 237.

Compton, K. and Hauck, S. 2002. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys 34,* 2 (June), 171–210.

D. De Waleffe and J.-J. Quisquater. 1990. CORSAIR: A smart card for public key cryptosystems. In *Advances in Cryptology - CRYPTO '90*, A. J. Menezes and S. A. Vanstone, Eds. Vol. LNCS 537. Springer-Verlag, Berlin, 502–514.

Dandalis, A., Prasanna, V. K., and Rolim, J. D. P. 2000a. A Comparative Study of Performance of AES Final Candidates Using FPGAs. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, Worcester, Massachusetts, USA, 125–140.

Dandalis, A., Prasanna, V. K., and Rolim, J. D. P. 2000b. An Adaptive Cryptographic Engine for IPSec Architectures. In *Eighth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '00*, K. L. Pocek and J. M. Arnold, Eds.

Davies, N. and Gellersen, H.-W. 2002. Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. *IEEE Pervasive Computing 1,* 1 (January–March), 26–35.

Dhem, J.-F. 1994. Modified version of the Barret modular multiplication algorithm. UCL Technical Report CG-1994/1, Université catholique de Louvain. July 18,.

Dhem, J.-F. 1998. Design of an efficient public-key cryptographic library for RISC-based smart cards. Ph.D. thesis, UCL — Université catholique de Louvain, Louvain-la-Neuve, Belgium.

Dierks, T. and Allen, C. 1999. *RFC 2246: The TLS Protocol Version 1.0.* Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA.

Diffie, W. and Hellman, M. E. 1976. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 644–654.

Dipert, B. 2000. Cunning circuits confound crooks. Available at `http://www.e-insite.net/ednmag/contents/images/21df2.pdf`.

Dussé, S. R. and Kaliski, B. S. 1990. A Cryptographic Library for the Motorola DSP56000. In *Advances in Cryptology — EUROCRYPT '90*, I. B. Damgård, Ed. Vol. LNCS 473. Springer-Verlag, Berlin, Germany, 230–244.

Elbirt, A. 1999. An FPGA Implementation and Performance Evaluation of the CAST-256 Block Cipher. Technical Report, Cryptography and Information Security Group, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA. May.

Elbirt, A. and Paar, C. 2000. An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher. In *FPGA '00 - ACM/SIGDA International Symposium on Field Programmable Gate Arrays.* ACM, Monterey, CA, USA, 33–40.

Elbirt, A. and Paar, C. 2001. An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. *IEEE Transactions on Very Large Integration (VLSI) Systems 4,* 9, 545–557.

Elbirt, A., Yip, W., Chetwynd, B., and Paar, C. 2000. An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. In *The Third Advanced Encryption Standard Candidate Conference.* National Institute of Standards and Technology, New York, New York, USA, 13–27.

Elbirt, A., Yip, W., Chetwynd, B., and Paar, C. 2001. An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. *IEEE Transactions on VLSI Design 9,* 4 (August), 545–557.

Eldridge, S. E. and Walter, C. D. 1993. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers 42,* 6 (July), 693–699.

Erickson, C. R. 1999. Configuration Stream Encryption. United States Patent, Patent Number 5970142.

Federal Information Processing Standards 1977. *NIST FIPS PUB 46, Data Encryption Standard.* Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce.

Ferreira, R., Malzahn, R., Marissen, P., Quisquater, J.-J., and Wille, T. 1996. FAME: A 3rd generation coprocessor for optimising public key cryptosystems in smart card applications. In *Proceedings of CARDIS 1996, Smart Card Research and Advanced Applications*, P. H. Hartel, P. Paradinas, and J.-J. Quisquater, Eds. Stichting Mathematisch Centrum, CWI, Amsterdam, The Netherlands, 59–72.

Fischer, V. and Drutarovsky, M. 2001. Two Methods of Rijndael Implementation in Reconfigurable Hardware. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. Vol. LNCS 2162. Springer-Verlag, 77–92.

Frecking, W. and Parhi, K. K. 1999. A unified method for iterative computation of modular multiplications and reduction operations. In *International Conference on Computer Design – ICCD '99.* 80–87.

Freier, A. O., Karlton, P., and Kocher, P. C. 1996. *The SSL Protocol Version 3.0.* Transport Layer Security Working Group INTERNET-DRAFT.

Gaj, K. and Chodowiec, P. 2000. Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware. In *The Third Advanced Encryption Standard Candidate Conference*. National Institute of Standards and Technology, New York, New York, USA, 40–54.

Gaj, K. and Chodowiec, P. 2001. Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays. In *Topics in Cryptology - CT-RSA 2001*, D. Naccache, Ed. Vol. LNCS 2020. Springer-Verlag, Berlin, 84 – 99.

Gordon, D. M. 1998. A survey of fast exponentiation methods. *Journal of Algorithms 27*, 129–146.

Goubin, L. and Patarin, J. 1999. DES and Differential Power Analysis. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 1999*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1717. Springer-Verlag, 158–172.

Guccione, S. A. and Levi, D. Jbits: A java-based interface to fpga hardware. Tech. rep., Xilinx Corporation, San Jose, CA, USA. Available at `http://www.io.com/~guccione/Papers/Papers.html`.

Gutmann, P. 1996. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Sixth USENIX Security Symposium*. 77–90.

Gutmann, P. 2001. Data Remanence in Semiconductor Devices. In *10th USENIX Security Symposium*. 39–54.

Haddad, S., Chang, C., Swaminathan, B., and Lien, J. 1989. Degradations due to hole trapping in flash memory cells. *IEEE Electron Device Letters 10,* 3 (March), 117–119.

Hauser, J. and Wawrzynek, J. 1997. Garp: A MIPS Processor with Reconfigurable Coprocessor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, K. Pocek and J. Arnold, Eds. 12–21.

Järvinen, K. U., Tommiska, M., and Skyttä, J. 2003. A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In *2003 ACM/SIGDA 11th International Symposium on Field programmable gate arrays — FPGA 2003*. ACM Press, 207–215.

Jeffrey, G. P. 2002. Field programmable gate arrays. United States Patent, Patent Number 6356637.

Kaliski, Jr., B. S., Koç, Ç. K., and Paar, C., Eds. 2002. *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*. Vol. LNCS 2523. Springer-Verlag, Berlin, Germany.

Kaps, J. P. 1998. High speed FPGA architectures for the Data Encryption Standard. M.S. thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA.

Kaps, J. P. and Paar, C. 1998. Fast DES implementation on FPGAs and its application to a universal key-search machine. In *Fifth Annual Workshop on Selected Areas in Cryptography*, S. Tavares and H. Meijer, Eds. Vol. LNCS 1556. Springer-Verlag, Berlin, Germany. Conference Location: Queen's University, Kingston, Ontario, Canada.

Kaps, J.-P. and Paar, C. 1999. DES auf FPGAs (DES on FPGAs, in German). *Datenschutz und Datensicherheit 23,* 10, 565–569. Invited contribution.

Kean, T. 2001. Secure Configuration of Field Programmable Gate Arrays. In *International Conference on Field-Programmable Logic and Applications 2001 (FPL 2001)*. Vol. LNCS 2147. Springer-Verlag, 142–151.

Kelem, S. H. and Burnham, J. L. 2000. System and Method for PLD Bitstram Encryption. United States Patent, Patent Number 6118868.

Kent, S. and Atkinson, R. 1998. *RFC 2401: Security Architecture for the Internet Protocol.* Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA.

Kessner, D. 2000. Copy Protection for SRAM based FPGA Designs. Available at `http://www.free-ip.com/copyprotection.html`.

Knuth, D. E. 1981. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, 2nd ed. Addison-Wesley, Reading, Massachusetts, USA.

Koblitz, N. 1987. Elliptic curve cryptosystems. *Mathematics of Computation 48*, 203–209.

Koç, Ç.. K. and Hung, C. Y. 1991. Bit-level systolic arrays for modular multiplication. *Journal of VLSI Signal Processing 3,* 3, 215–223.

Koç, Ç. K., Naccache, D., and Paar, C., Eds. 2001. *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2001*. Vol. LNCS 2162. Springer-Verlag, Berlin, Germany.

Koç, Ç. K. and Paar, C., Eds. 1999. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'99*. Vol. LNCS 1717. Springer-Verlag, Berlin, Germany.

Koç, Ç. K. and Paar, C., Eds. 2000. *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*. Vol. LNCS 1965. Springer-Verlag, Berlin, Germany.

Kocher, P., Jaffe, J., and Jun, B. 1999. Differential Power Analysis. In *Advances in Cryptology — CRYPTO '99*, M. Wiener, Ed. Vol. LNCS 1666. Springer-Verlag, 388–397.

Kommerling, O. and Kuhn, M. 1999. Design Principles for Tamper-Resistant Smartcard Processors. In *USENIX Workshop on Smartcard Technology (Smartcard '99)*. 9–20.

Kornerup, P. 1994. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Transactions on Computers 43,* 8 (August), 892–898.

Lai, X. and Massey, J. 1990. A proposal for a new block encryption standard. In *Advances in Cryptology — EUROCRYPT '90*, I. B. Damgård, Ed. Vol. LNCS 473. Springer-Verlag, Berlin, Germany, 389–404.

Lai, X. and Massey, J. L. 1991. Markov Ciphers and Differential Cryptanalysis. In *Advances in Cryptology — EUROCRYPT '91*, D. W. Davies, Ed. Vol. LNCS 547. Springer-Verlag, Berlin, Germany, 17–38.

Lai, X., Massey, Y., and Murphy, S. 1991. Markov Ciphers and Differential Cryptoanalysis. In *Advances in Cryptology — EUROCRYPT '91*, D. W. Davies, Ed. Vol. LNCS 547. Springer-Verlag, Berlin, Germany.

Lenstra, A. and Verheul, E. 2001. Selecting cryptographic key sizes. *Journal of Cryptology 14,* 4, 255–293.

Leonard, J. and Magione-Smith, W. 1997. A case study of partially evaluated hardware circuits: Keyspecific DES. In *Field-Programmable Logic and Applications, 7th International Workshop, FPL '97*, W. Luk, P. Cheung, and M. Glesner, Eds. Springer-Verlag, London, UK.

Lipmaa, H. 2002. Fast Software Implementations of AES. Available at `http://www.tcs.hut.fi/~helger/aes/rijndael.html`.

M. Dworkin. 2001. *NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation — Methods and Techniques.* National Institute of Standards and Technology/U.S. Department of Commerce. Available at `http://csrc.nist.gov/encryption/tkmodes.html`.

M. Dworkin. 2002. *Draft NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The RMAC Authentication Mode — Methods and Techniques.* National Institute of Standards and Technology/U.S. Department of Commerce. Available at `http://csrc.nist.gov/encryption/tkmodes.html`.

Massey, J. L. and Lai, X. 1992. Device for converting a digital block and the use thereof. European Patent, Patent Number 482154.

McLoone, M. and McCanny, J. 2001. High Performance Single-Chip FPGA Rijndael Algorithm. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. Vol. LNCS 2162. Springer-Verlag, 65–76.

Menezes, A. and Johnson, D. 1999. The elliptic curve digitial signature algorithm (ECDSA). Technical report CORR 99-34, Department of C & O, University of Waterloo, Ontario, Canada. August.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. 1997. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, Florida, USA.

Miller, V. 1986a. Uses of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85*, H. C. WIlliams, Ed. Vol. LNCS 218. Springer-Verlag, Berlin, Germany, 417–426.

Miller, V. 1986b. Uses of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85*, H. C. Williams, Ed. Vol. LNCS 218. Springer-Verlag, Berlin, Germany, 417–426.

Montgomery, P. L. 1985. Modular multiplication without trial division. *Mathematics of Computation 44,* 170 (April), 519–521.

Naccache, D. and M'Raïhi, D. 1996. Cryptographic smart cards. *IEEE Micro 16,* 3, 14–24.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E. 2000. Report on the Development of the Adavanced Encryption Standard (AES). Available at csrc.nist.gov/encryption/aes/round2/r2report.pdf, National Institute of Standards and Technology/U.S. Department of Commerce. October 2,,.

Norris, M. J. and Simmons, G. J. 1981. Algorithms for high-speed modular arithmetic. *Congressus Numeratium 31*, 153–163.

Omura, J. K. 1990. A public key cell design for smart card chips. In *International Symposium on Information Theory and its Applications*. USA, 983–985.

Orlando, G. and Paar, C. 2001. A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. Vol. LNCS 2162. Springer-Verlag, 348–363.

Örs, S., Oswald, E., and Preneel, B. 2003. Power-Analysis Attacks on an FPGA — First Experimental Results. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, C. Walter, Ç. K. Koç, and C. Paar, Eds. Vol. LNCS 2779. Springer-Verlag, 35–50.

Orup, H. 1995. Simplifying quotient determination in high-radix modular multiplication. In *Proceedigns of the 12th IEEE Symposium on Computer Arithmetic (ARITH 12)*. 193–9.

Orup, H. and Kornerup, P. 1991. A High-Radix Hardware Algorithm for Calculating the Exponential M^E Modulo N. In *Proceedigns of the 10th IEEE Symposium on Computer Arithmetic (ARITH 10)*, P. Kornerup and D. W. Matula, Eds. 51–56.

P1363 2000. *IEEE P1363-2000: IEEE Standard Specifications for Public Key Cryptography.* Available at http://standards.ieee.org/catalog/olis/busarch.html.

Pang, R. C., Wong, J., Frake, S. O., Sowards, J. W., Kondapalli, V. M., Goetting, F. E., Trimberger, S. M., and Rao, K. K. 2000. Nonvolatile/battery-backed key in PLD. United States Patent, Patent Number 6366117.

Papadas, C., Ghibaudo, G., Pananakakis, G., Riva, C., Ghezzi, P., Gounelle, C., and Mortini, P. 1991. Retention characteristics of single-poly EEPROM cells. In *European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.* 517.

Patterson, C. 2000a. A Dynamic Implementation of the Serpent Block Cipher. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000)*, Çetin K. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, 142–156.

Patterson, C. 2000b. High Performance DES Encryption in Virtex FPGAs Using JBits. In *IEEE Symposium on Field–Programmable Custom Computing Machines (FCCM 2000)*, K. L. Pocek and J. M. Arnold, Eds.

Preneel, B., Van Rompay, B., Örs, S., Biryukov, A., Granboulan, L., Dottax, E., Dichtl, M., Schafheutle, M., Serf, P., Pyka, S., Biham, E., Barkan, E., Dunkelman, O., Stolin, J., Ciet, M., Quisquater, J.-J., Sica, F., Raddum, H., and Parker, M. 2003. Performance of Optimized Implementations of the NESSIE Primitives. Tech. rep. February 20. Available at http://www.cryptonessie.org/.

Quisquater, J.-J. Fast modular exponentiation without division. Rump session of EUROCRYPT '90.

Quisquater, J.-J. 1992. Encoding system according to the so-called RSA method, by means of a microcontroller and arrangement implementing this system. United States Patent, Patent Number 5166978.

Quisquater, J.-J. and Couvreur, C. 1982. Fast decipherment algorithm for RSA public–key cryptosystem. *Electronics Letters 18*, 905–907.

Quisquater, J.-J. and Samyde, D. 2001. Electro Magnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In *International Conference on Research in Smart Cards, E-smart 2001.* Cannes, France, 200 – 210.

Rashid, A., Leonard, J., and Mangione-Smith, W. 1998. Dynamic circuit generation for solving specific problem instances of boolean satisfiability. In *IEEE Symposium on FPGAs for Custom Computing Machines - FCCM '98.* Napa Valley, California, USA, 196–205.

Riaz, M. and Heys, H. 1999. The FPGA Implementation of RC6 and CAST-256 Encryption Algorithms. In *Proceedings: IEEE 1999 Canadian Conference on Electrical and Computer Engineering*. Edmonton, Alberta, Canada.

Richard, G. 1998. Digital Signature Technology Aids IP Protection. In EETimes - News. Available at `http://www.eetimes.com/news/98/1000news/digital.html`.

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

San, K., Kaya, C., and Ma, T. 1995. Effects of erase source bias on Flash EPROM device reliability. *IEEE Transactions on Electron Devices 42,* 1 (January), 150–159.

Schaumont, P., Verbauwhede, I., Keutzer, K., and Sarrafzadeh, M. 2001. A Quick Safari Through the Reconfiguration Jungle. In *Proceedings of the 38th Conference on Design Automation — DAC 2001*. ACM Press, New York, NY, USA, 172–177.

Schneier, B. 1996. *Applied Cryptography*, 2nd ed. John Wiley & Sons Inc., New York, New York, USA.

Schroder, D. 1998. *Semiconducor Material and Device Characterization*, 2nd ed. John Wiley and Sons.

Seamann, G. 2000. FPGA Bitstreams and Open Designs. Available at `http://www.opencollector.org/news/Bitstream`.

Sedlak, H. 1987. The rsa cryptography processor. In *Advances in Cryptology – EUROCRYPT '87*, D. Chaum and W. L. Price, Eds. Vol. LNCS 304. Springer-Verlag, Berlin, Germany, 95–105.

Shamir, A. 2000. Protecting Smart Cards form Power Analysis with Detached Power Supplies. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, 71–77.

Shand, M. and Vuillemin, J. 1993. Fast implementations of RSA cryptography. In *Proceedigns of the 11th IEEE Symposium on Computer Arithmetic (ARITH-11)*, E. Swartzlander, Jr., M. J. Irwin, and G. Jullien, Eds. 252–259.

Shang, L., Kaviani, A., and Bathala, K. 2002. Dynamic Power Consumption on the Virtex-II FPGA Family. In *2002 ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays*. ACM Press, 157–164.

Skorobogatov, S. and Anderson, R. 2002. Optical Fault Induction Attacks. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, Eds. Vol. LNCS 2523. Springer-Verlag, 2–12.

Soden, J. and Anderson, R. 1993. IC failure analysis: techniques and tools for quality and reliability improvement. *Proceedings of the IEEE 81,* 5 (May), 703–715.

Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., and Legat, J.-D. 2003a. A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL. In *2003 ACM/SIGDA 11th International Symposium on Field programmable gate arrays — FPGA 2003*. ACM Press, 216–224.

Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., and Legat, J.-D. 2003b. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, C. Walter, Ç. K. Koç, and C. Paar, Eds. Vol. LNCS 2779. Springer-Verlag, 334–350.

Standaert, F.-X., van Oldeneel tot Oldenzeel, L., Samyde, D., and Quisquater, J.-J. 2003. Power Analysis of FPGAs: How Practical is the Attack. In *13th International Conference on Field Programmable Logic and Applications — FPL 2003*, P. Cheung, G. Constantinides, and J. de Sousa, Eds. Vol. LNCS 2778. Springer-Verlag.

Sung, C. and Wang, B. I. 1999. Method and Apparatus for Securing Programming Data of Programmable Logic Device. United States Patent, Patent Number 5970142.

Tao, J., Cheung, N., and Ho, C. 1993. Metal Electromigration Damage Healing Under Bidirectional Current Stress. *IEEE Transactions on Elecron Devices 14,* 12 (December), 554–556.

Taylor, R. and Goldstein, S. 1999. A high-performance flexible architecture for cryptography. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES '99*, Ç. Koç and C. Paar, Eds. Vol. LNCS 1717. Springer-Verlag, Worcester, Massachusetts, USA, 231–245.

Tessier, R. and Burleson, W. 2000. Reconfigurable Computing for Digital Signal Processing: A Survey. *Journal of VLSI Signal Processing 28,* 1 (June), 7–27.

Thomas, S., Anthony, D., Berson, T., and Gong, G. 2003. The W7 Stream Cipher Algorithm. Available at `http://www.watersprings.org/pub/id/draft-thomas-w7cipher-03.txt`. Internet Draft.

Trimberger, S., Pang, R., and Singh, A. 2000. A 12 Gbps DES Encryptor/Decryptor Core in an FPGA. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1965. Springer-Verlag, Worcester, Massachusetts, USA, 157–163.

Triscend Corporation. Available at `http://www.triscend.com/`.

U.S. Department of Commerce/National Institute of Standard and Technology 2000b. *FIPS 186-2, Digital Signature Standard (DSS).* U.S. Department of Commerce/National Institute of Standard and Technology. Available at `http://csrc.nist.gov/encryption`.

U.S. Department of Commerce/National Institute of Standard and Technology 2001a. *FIPS PUB 197, Specification for the Advanced Encryption Standard (AES).* U.S. Department of Commerce/National Institute of Standard and Technology. Available at `http://csrc.nist.gov/encryption/aes`.

U.S. Department of Commerce/National Institute of Standards and Technology 1999. *NIST FIPS PUB 46-3, Data Encryption Standard (DES).* U.S. Department of Commerce/National Institute of Standards and Technology. Available at `http://csrc.nist.gov/encryption/tkencryption.html`.

van der Pol, J. and Koomen, J. 1990. Relation between the hot carrier lifetime of transistors and CMOS SRAM products. In *International Reliability Physics Symposium (IRPS 1990).* 178.

Vuillemin, J. E., Bertin, P., Roncin, D., Shand, M., Touati, H. H., and Boucard, P. 1996. Programmable active memories: Reconfigurable systems come of age. *IEEE Transactions on VLSI Systems 4,* 1 (March), 56–69.

Walter, C. D. 1991. Faster modular multiplication by operand scaling. In *Advances in Cryptology - CRYPTO '91*, J. Feigenbaum, Ed. Vol. LNCS 576. Springer-Verlag, Berlin, 313–323.

Weaver, N. and Wawrzynek, J. 2000. A Comparison of the AES Candidates Amenability to FPGA Implemenation. In *The Third Advanced Encryption Standard Candidate Conference.* National Institute of Standards and Technology, New York, New York, USA, 28–39.

Wilcox, D. C., Pierson, L., Robertson, P., Witzke, E., and Gass, K. 1999. A DES ASIC Suitable for Network Encryption at 10 Gbps and Beyond. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES '99*, Ç. Koç and C. Paar, Eds. Vol. LNCS 1717. Springer-Verlag, Worcester, Massachusetts, USA, 37–48.

Williams, T., Kapur, R., Mercer, M., Dennard, R., and Maly, W. 1996. IDDQ Testing for High Performance CMOS - The Next Ten Years. In *IEEE European Design and Test Conference (ED&TC'96).* 578–583.

Wollinger, T., Wang, M., Guajardo, J., and Paar, C. 2000. How well are high-end DSPs suited for the AES algorithms? In *The Third Advanced Encryption Standard Candidate Conference.* National Institute of Standards and Technology, New York, New York, USA, 94–105.

Wong, S., Vassiliadis, S., and Cotofana, S. 2002. Future Directions of (Programmable and Reconfigurable) Embedded Processors. In *Embedded Processor Design Challenges, Workshop on Systems, Architectures, Modeling, and Simulation — SAMOS 2002.*

Wu, H. 1999. Low complexity bit-parallel finite field arithmetic using polynomial basis. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 1999*, Ç. K. Koç and C. Paar, Eds. Vol. LNCS 1717. Springer-Verlag, 280 – 291.

Xilinx Inc. Using Bitstream Encryption. Handbook of the Virtex II Platform. Available at `http://www.xilinx.com`.

Xilinx Inc. 1999. *XC4000E and XC4000X Series Field Programmable Gate Arrays (Version 1.6).* Xilinx Inc., San Jose, California, USA.

Xilinx Inc. 2001. *Virtex 2.5V Field Programmable Gate Arrays (Version 2.5).* Xilinx Inc., San Jose, California, USA.

Xilinx Inc. 2002. *Virtex^{TM}-II Platform FPGA Data Sheet.* Xilinx Inc. Available at `http://www.xilinx.com/partinfo/databook.htm`.

Xilinx Inc. 2003. *Virtex-II Pro^{TM} Platform FPGAs: Introduction and Overview.* Xilinx Inc. Version 2.4. Available at `http://direct.xilinx.com/bvdocs/publications/ds083.pdf`.

Yip, K.-W. and Ng, T.-S. 2000. Partial-Encryption Technique for Intellectual Property Protection of FPGA-based Products. *IEEE Transactions on Consumer Electronics 46,* 1, 183–190.

APPENDIX

A. BACKGROUND: PUBLIC-KEY AND SYMMETRIC-KEY ALGORITHMS

A.1 Symmetric-key Algorithms

Private-key or Symmetric-key algorithms are algorithms where the encryption and decryption key is the same and which encrypt the data, often at high speeds. Private-key algorithms require the sender and the receiver to agree on the key prior to the communication taking place and the security rests in this key. There are two types of symmetric-key algorithms which are commonly distinguished: block ciphers and stream ciphers [Schneier 1996].

Block ciphers encrypted the message broken into blocks of fixed length and examples include Data Encryption Standard (DES) [Federal Information Processing Standards 1977], the International Encryption Standard (IDEA) [Lai and Massey 1991; Massey and Lai 1992], and the Advanced Encryption Standard (AES) [U.S. Department of Commerce/National Institute of Standard and Technology 2001a]. AES became a US standard in October 2000 and supports block size of 128 bits and 128-bit, 192-bit, and 256-bit long keys. It is important to point out that the trend in modern symmetric-key cipher design has been to optimize the algorithms for efficient software implementations in contrast to DES which was designed with hardware in mind.

In practical applications, if one needs to encrypt larger blocks than the block size of the block-cipher the data will be partitioned into blocks of size n, where n is the input size of the block-cipher[6]. Each block is encrypted via different methods or so called "modes of operation", like Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter mode [Menezes et al. 1997; M. Dworkin 2001; 2002]. Considering modes of operation is important because depending on the mode different methodologies might be used to enhance the throughput of a block cipher.

Stream ciphers operate on a single bit of plaintext at a time. They are useful because the encryption transformation can change for each symbol of the message being encrypted. In particular, they are useful in situations where transmission errors are highly probable because they do not have error propagation. In addition, they can be used when the data must be processed one symbol at a time because of lack of equipment memory or limited buffering.

As a final remark, notice that one of the major issues with symmetric-key systems is the need to find an efficient method to agree on and exchange the secret keys securely [Menezes et al. 1997]. In [1976], Diffie and Hellman proposed a new concept

[6] We will use n to denote the block-size of the block cipher in the remaining of this contribution whenever we are discussing symmetric ciphers.

that would revolutionize cryptography as it was known at the time, called public-key cryptography.

A.2 Public-key Algorithms

Public-key (PK) cryptography is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. Anyone that wants to send a message to party A can encrypt that message using A's *public key* but only A can decrypt the message using her *private key.* It is understood that A's private key should be kept secret at all times and A's public key is publicly available to everyone. Furthermore, it is impossible for anyone, except A, to derive the private key (or at least to do so in any reasonable amount of time).

In general, one can divide practical public-key algorithms into three families:

—Algorithms based on the *integer factorization problem:* given a positive integer n, find its prime factorization, e.g. RSA [Rivest et al. 1978].

—Algorithms based on the *discrete logarithm problem:* given α and β find x such that $\beta = \alpha^x \bmod p$, including the Diffie-Hellman [Diffie and Hellman 1976] key exchange protocol and the Digital Signature Algorithm (DSA).

—Algorithms based on *Elliptic Curves.* Elliptic curve cryptosystems [Miller 1986b; Koblitz 1987] are the most recent family of practical public-key algorithms, which have gained acceptance including standardization [P1363 2000].

Despite the differences between these mathematical problems, all three algorithm families have something in common: they all perform complex operations on very large numbers, typically 1024–2048 bits in length for the RSA and discrete logarithm systems or 160–256 bits in length for the elliptic curve systems[7]. PK systems have a major disadvantage, they are very arithmetic intensive and even when properly implemented, all PK schemes proposed to date are several orders of magnitude slower than the best known private-key schemes.

B. FPGA TECHNOLOGY

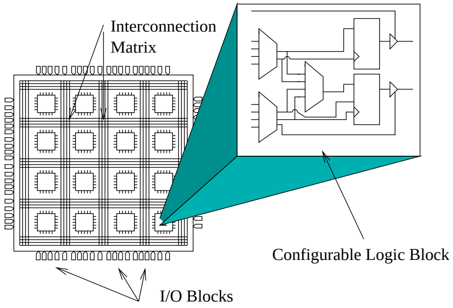
The Xilinx Virtex family is the most used FPGA series in academia concerning cryptographic implementations, as it can be seen from the summaries of results in Section 6 and Section 7. This section will give some more detailed description, about FPGA in general and the chips used in the cited contributions.

The original 2.5-Volt Virtex family was introduced in 1998 offering features, like Block RAM, Distributed RAM and High-speed external memory interfaces, Delay-Locked Loops (DLLs), and SelectI/O. The Virtex-E family, introduced in 1999, delivers more RAM, more DLLs, the SelectLink^{TM} technology and high-speed differential signaling. Virtex-II and Virtex-II Pro FPGAs are the high end chips offered by Xilinx.

One of the biggest architectural differences between FPGAs and CPLDs is that FPGAs have an array of many small logic blocks with vast interconnection networks, while Complex Logic Device (CPLDs) have a few large logic block based on PALs, with smaller interconnection networks. Hence, FPGAs exist of three main

[7] We refer to [Lenstra and Verheul 2001] for extended discussions regarding key equivalences between different asymmetric and symmetric cryptosystems.

Fig. 1. 2-Slice Virtex CLB



components: Configurable Logic Blocks (CLBs), interconnections, and I/O blocks (see Figure 1).

FPGA technology is usually based on SRAM, flash, EEPROM or anti-fuse interconnections. The Virtex family is based on SRAM technology. The I/O blocks of FPGAs are very similar to the I/O pads in an ASIC and act as buffers to the outside world. The CLBs are the core logic element in an FPGA. The main block in a Virtex CLBs is the logic cell (LC). Each Virtex CLB contains of four LCs, organized in two similar slices. An LC includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop.

The slice includes 4-input look-up tables (LUTs), which are the function generators of the CLB. Each LUT can provide a 16 x 1-bit synchronous RAM and the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16x1-bit dual-port synchronous RAM. The F5 multiplexer provides the ability to combine the function generator outputs, either to a function generator (implementing any 5-input function), to a 4:1 multiplexer, or to a selected functions of up to nine inputs. F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions of up to 19 inputs. The XOR gate provides the possibility to implement a 1-bit full adder in one LC and the AND gate allows a efficient multiplier implementation.

Moreover, large block of RAM memories which are organized in columns are provided. Virtex devices have two columns that extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Virtex device 64 CLBs high contains 16 memory blocks per column, and a total of 32 blocks.

A summary of the number of system gates, CLBs, LC, available I/O and RAM for the Virtex devices can be found in Table VII.

Besides the Virtex family the XC4000 devices were used in some mentioned implementations. The XC4000 family was introduced in 1992 as Xilinxs third generation and spans from 3,000 to 125,000 gates. XC4000 CLB consists of two 4-input LUTs, one 3-input LUT, and two flip-flops. For example the XC4025 FPGA consists of 1024 CLBs, 2560 Flip-Flops and 256 I/O pins.

Table VII. Virtex FPGA Family Members

Device	System Gates	CLB Array	Logic Cells	Maximum I/O	Block RAM Bits	Maximum RAM Bits
XCV50	57,906	16x24	1,728	180	32,768	24,576
XCV100	108,904	20x30	2,700	180	40,960	38,400
XCV150	164,674	24x36	3,888	260	49,152	55,296
XCV200	236,666	28x42	5,292	284	57,344	75,264
XCV300	322,970	32x48	6,912	316	65,536	98,304
XCV400	468,252	40x60	10,800	404	81,920	153,600
XCV600	661,111	48x72	15,552	512	98,304	221,184
XCV800	888,439	56x84	21,168	512	114,688	301,056
XCV1000	1,124,022	64x96	27,648	512	131,072	393,216