Authentication of FPGA Bitstreams: Why and How

Saar Drimer*

Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, UK
saar.drimer@cl.cam.ac.uk

Abstract. Encryption of volatile FPGA bitstreams provides confidentiality to the design but does not ensure its authenticity. This paper motivates the need for adding authentication to the configuration process by providing application examples where this functionality would be useful. An examination of possible solutions is followed by suggesting a practical one in consideration of the FPGA's configuration environment constraints. The solution proposed here involves two symmetric-key encryption cores running in parallel to provide both authentication and confidentiality while sharing resources for efficient implementation.

1 Introduction

The most significant security shortcoming of volatile *Field Programmable Gate Arrays* (FPGA) is the transmission, on every power-up, of their functional definition, the bitstream, from an off-chip source. Field update is what FPGAs are designed for and is one of their distinct advantages, but also what exposes them to various attacks. Some FPGAs now have the ability to process encrypted bitstreams in order to maintain the design's confidentiality. Authenticity, however, is not provided and results in the FPGA accepting bitstreams of *any* form and from *anyone* with the consequence of, at best, denial of service, or at worst, the execution of unauthorized code.

This paper describes the benefits of adding bitstream authentication to the configuration process, a topic that has only been superficially covered to date. While previously suggested, a thorough discussion about the particulars has not been made. The goal of the paper is to motivate the topic and to propose a solution that considers all constraints in order for it to be appealing enough for integration into commercial FPGAs. The paper starts with a short introduction to relevant security goals, followed by examples of real world applications that would benefit from authentication. After considering available and previously suggested schemes, the proposed solution is described.

* This work has been supported by a grant from Xilinx Inc., 2100 Logic Dr., San Jose, CA 95124, USA.

2 The Case for Authentication

Confidentiality is typically preserved by encryption—an invertible non-linear function that relies on a secret: a *key*. The most common form of encryption uses *symmetric ciphers* where the two parties communicating share a pre-arranged key. This allows the data to be transmitted on an insecure channel as either party encrypts or decrypts it at each end. The *Advanced Encryption Standard* (AES) became the most commonly used symmetric cipher, after NIST chose to replace the *Data Encryption Standard* (DES) in 2001 [1]. For our purposes, it is sufficient to know that AES processes data in 10 rounds, each needing a sub-key that is derived from the main key. The element that contributes the most to the circuit size is the *substitution box* (s-box), which simply converts an n bit input value to a different n bit value. These s-boxes are part of both the main round-function and the key-scheduling function and are typically implemented as look-up tables.

Public-Key Cryptography (PKC) is a different, much more computationally demanding protocol, where the parties are not required to agree on mutual keys prior to communicating. The process is slower than symmetric ciphers and requires longer keys for equivalent security. PKC is most often used for producing digital signatures and establishing symmetric session keys during the initial stages of secure communication.

Authentication provides cryptographic-level assurance of data integrity and its source. Hash functions process arbitrary-length data and produce a fixed-length string called a *message digest*, *hash*, or *tag*. When the process incorporates a secret key, the outcome is called a *Message Authentication Code* (MAC) and allows the receiver to verify that the message is authentic: it has not been tampered with and the sender knows the key. This family of algorithms is designed such that it is computationally infeasible to find two different inputs that result in identical MACs, find an input that yields a given MAC, or find a MAC for a given input without knowing the key.

"In general, authentication is more important than encryption" conclude Schneier and Ferguson in [2, pp. 116]. Being able to impersonate someone to provide falsified information can be more devastating than just being able to eavesdrop on communications. This applies to secure communications in general and emphasizes the importance of authentication in many applications.

For bitstreams, authentication allows the FPGA to verify that the bitstream's content it is about execute was created by someone with whom it shares a secret and that it has not been altered in any way. Doesn't encryption provide the same assurance? As we shall see, the answer is no. Each function should be considered on its own, as each serves a different purpose that sometimes does not require the other. In essence, encryption protects the bitstream's content independently of the device (from cloning, reverse engineering, etc.) while authentication ensures the *correct* and *intended* operation of the FPGA.

The rest of this section describes applications where authentication plays a crucial role in the overall correct and secure operation of the system. To illustrate these, a fictitious company is formed, "Honest Voting Machines, Inc." (HVM)

which is set on producing verifiably honest voting machines. The field upgradability and agility of FPGAs appeal to them, and they cannot justify the design of an ASIC. They are looking at ways in which they can achieve their security goals while still being able to use an FPGA as their main processor.

2.1 Data Integrity

HVM is considering using encryption for their bitstreams. They are aware, though, that encryption alone does not provide them with data integrity since data can be manipulated without knowledge of the key. Symmetric ciphers have several *modes of operation* that define how data is processed [3]. The various modes allow the core function of a symmetric cipher to operate as a block cipher, stream cipher or even produce a MAC. Each of these modes have respective security properties that must be evaluated per the application. A popular mode is *Cipher Block Chaining* (CBC) that is used to encrypt arbitrary-length messages such that ciphertext blocks are not the same for identical plaintext blocks. CBC is self recovering in that it can recover from ciphertext bit errors within two decryption operations. If m bits are manipulated in a block of size n, $n/2$ incorrect bits, on average, will be incorrect in the resulting plaintext block. An additional m bits will be flipped in the subsequent plaintext block at the exact location of the original manipulated bits. When using CBC for bitstream encryption, an adversary can toggle m bits in a single block to change the design without knowing the key. The previous block will be completely corrupt but it may not contain data that is relevant to the attacked portion of the chip. This attack can be fruitful, for example, if the goal is to find a collection of bits that, when toggled, disable a defence mechanism. Other modes have similar, or worse, vulnerabilities with regards to manipulation of ciphertexts.

Hadžić et al. were the first to tackle the implication of malicious tampering of bitstreams [4]. The paper describes how an FPGA is permanently damaged due to shorts caused by manipulating configuration bits that control pass gates. The high currents caused by these shorts may cause the device to exceed its thermal tolerance, permanently damaging it. A bitstream of random content will create considerable contention and is relatively simple to construct; if one knows the bitstream construction, an even more devious one can be made. This attack, however, is not at all likely to succeed for most fielded systems. To damage the FPGA, the power supplies need be able to provide the high current needed for sufficiently long periods while the supporting circuitry be able to handle that current. This is not the case in most systems where the voltage regulators collapse upon high current demand, or some other failures occur, long before the device can be damaged. In addition, FPGAs have circuitry that detects voltage drops, usually 15% below nominal, in which case, they reset themselves. Although not suggested in [4], authentication solves all issues presented there.

Error correction (and detection) codes, such as *Cyclic Redundancy Codes* (CRC) do not satisfy HVM either. These codes are currently used to prevent a bitstream that was accidentally corrupted in transmission from being loaded onto the device and potentially causing damage due to circuit shorts. These

linear functions can be forged with relative ease and do not detect all types of errors, especially ones put there maliciously [5]. Linear checks are therefore not adequate for cryptographic-level data integrity. Authentication protocols provide this level of assurance by being designed such that even a single wrong bit will result in an incorrect MAC.

2.2 Code Audit

HVM has been contracted by a large municipality to provide it with voting machines. The municipality would like to assure itself, and its constituents, that the code running on the FPGA-based system does not have back-doors or election-altering functions. HVM, in turn, insists that only their proprietary code is run on their voting machines. Those interests are met as follows.

HVM provides the source code to the municipality, or posts it on its web site, along with a MAC for the compiled design. That MAC was computed using a key that HVM embeds into the FPGA that will be part of the system sold to the municipality. HVM also provides the software version and settings such that the code would compile to exactly the same bitstream every time, provided that there are no design changes. Given that information, the municipality compiles the verified code to produce a bitstream to which they append the MAC that was provided by HVM. The resultant bitstream is then loaded onto the FPGA by the municipality. Only HVM knows the key to compute a valid MAC and therefore, no-one else can create a bitstream that will run on that particular FPGA. Since the design was not encrypted and the code verified, the municipality is assured of its correctness.

2.3 Versioning of Identical Hardware

It makes economic sense for HVM to design hardware that can function, depending on the bitstream, in "standard" or "enhanced" modes, the latter having more functionality and is more expensive. HVM embeds the MAC key to either version in each system's FPGA. The code audit scheme allows for the code for both variants to be publicly available, yet would not let the customers replace the standard bitstream with the enhanced one.

This and the previous scheme require that the MAC key be permanently embedded within the FPGA, otherwise it could be erased and the system used with the publicly available, or other, code. Some FPGAs already allow permanent key storage for encryption; storage for the authentication key will need to be provided as well. It also requires strict inventory control by both seller and customer in order to prevent counterfeit products from being manufactured or purchased with the public design modified for malicious intent.

2.4 Authenticated Heart Beat

Consider a critical system that is operating in an insecure and remote environment. For example, HVM's voting machine during an election. The goal is to

provide to both the code running on the FPGA and a remote monitoring center (MC) a guarantee that they are continuously connected to one another. If either the code or the MC detects a lapse in connectivity, a breach is assumed. An internal authentication core with a unique key supplemented by a volatile counter can accomplish this.

On election day, officials load the voting machine with the previously audited code along with a predetermined initial state of the counter, C_i. Once configuration is complete, an initial authentication process is performed as follows.

$$A \rightarrow B : N_A \in \{0,1\}^{128}, \mathrm{H}_K(N_A||C_{i+1})$$
$$B \rightarrow A : \mathrm{H}_K(N_A||C_{i+2})$$

A, the monitoring center, sends B, the FPGA, a nonce N_A and a MAC, H_K, of the nonce concatenated with the current value of C under the key, K. The FPGA computes the MAC on its end and stores the result in a status bit that is constantly monitored by the design. The FPGA then replies with a MAC computed with C_{i+2} and N_A. The authentication process repeats in the background as often as needed, each time with a unique nonce from the MC an incremented counter value, C_i. The counter and nonce provide freshness; the nonce should never be reused to assure that previously recorded challenges can not be used in the future by an attacker. The authentication process does not interfere with the normal operation of the device.

3 Constraints

Now that some of the benefits of authentication have been described, the constraints that bound a solution will be considered for arriving at an optimal solution.

3.1 Configuration Environment

Systems deal with remote and local configuration of an FPGA in a variety of ways. Some have non-volatile memory placed close to the FPGA, some configure via a microcontroller, while in others, the configuration is fetched, on every power-up, via a network or a local PC. Authentication, or other bitstream pre-processing, can be done using devices external to the FPGA, such as microcontrollers or dedicated processors, but only if the trust is set around the system itself. If this is the case, elaborate tamper proofing is necessary, or that the system be placed in a secure environment. This is not feasible for many, or most, applications. Thus, in this paper, the boundary of trust is set around the FPGA itself without delegating the bitstream pre-processing to other devices. Whether the adversary has physical access to the FPGA or not, malicious code must be prevented from being executed. Incorporating the authentication process into the FPGA accounts for bitstreams arriving by any means, be they from

a local non-volatile device, USB, ethernet, or the various other sources. This requirement means that implementing the authentication protocol in the FPGA's user logic will not work since an initial, trusted, bitstream must be loaded first. Since this is what we are trying to achieve to begin with, we must rely on a core that is part of the configuration process. Further, the requirement excludes verification mechanisms that rely on external devices and the continued secrecy of the bitstream's proprietary encoding, as suggested in [6,7] and the recently announced "Device DNA" from Xilinx [8].

Current FPGAs can only hold a single configuration data at a time. Once a new configuration process has begun, the previous one can not be reverted back to without full reconfiguration. No history is kept of previous failed attempts. Since we put the trust boundary around the FPGA, there is no way to guarantee that the *same* bitstream will be provided twice from an external source. Therefore, authentication and encryption operations must be done within the same configuration cycle.

Denial-of-service attacks can not be prevented with today's configuration process. If the attacker has physical access to the device he can destroy or unplug it. Short of effective tamper-proofing of the whole system, or placing it in a secure environment, there is not much to be done against a destructive attacker. If the attacker is remote, and has the ability to reconfigure the device, he can initiate a failed configuration attempt to leave the device un-configured. Authentication will prevent unauthorized configuration from running, but not denial-of-service. In addition, there is no way to prevent *replay attacks* where the adversary records a valid bitstream and sends it at a later time, perhaps to revert back to a previous bitstream with known vulnerabilities. All these issues stem from the fact that the FPGA does not preserve state between configurations.

Table 1. Configuration throughput and time of largest FPGA family members

FPGA	device	config bits	mode	frequency	throughput	time
Virtex-5 [9]	LX330T	82,696,192	8 bit[a]	100 MHz	800 Mbits/s	0.1 s
Stratix-III [10]	L340	125,829,120	8 bit	25 MHz[b]	200 Mbits/s	0.63 s
Spartan-3 [11]	5000	13,271,936	8 bit	50 MHz	400 Mbits/s	0.033 s

[a] Longer bus width, x16 and x32, are not supported when encryption is used
[b] Actual is 100 MHz; data must be x4 slower than clock if decryption is used

The adopted solution must meet the current configuration throughput and not adversely affect total configuration times. Table 1 shows the maximum throughput and configuration times of the largest family members of recent FPGAs.

The throughput of the authentication core needs to meet the maximum throughput of the decryption core, since they most likely operate together. According to Table 1, this is set around 800 Mbit/s, which is a modest requirement. Typically, as the throughput of a circuit increases, so does its size. It is impor-

tant that the core be designed such that it does not have greater throughput than required, so not to "waste" resources.

The effect of longer configuration times depends on the application and the mode. Some applications configure in serial mode at low frequencies, and therefore, doubling the configuration times may be undesirable. Applications that require frequent scrubbing to mitigate *Single Event Upsets*, for example, would suffer from longer configuration times. In general, therefore, it is best to find a solution that does not adversely affect the total time of configuration. That said, users may need to accept extra latency for the added functionality.

3.2 Manufacturers

Economical considerations should be mentioned as they play a crucial role when a new feature or technology is evaluated or considered for adoption. If the goal is to suggest something that would be considered to be viable, these constraints must be accounted for. From the perspective of the FPGA manufacturers, incorporating new functionality into their devices must yield a return on their investment. This means that either the functionality is required or demanded by sufficient amount customers, or, it is small and does not adversely affect performance, yield, support and development costs. Standardization is important as well as manufacturers prefer the use of "approved" algorithms that will be accepted by the majority of their customers. In the case of security, this is doubly important, as the manufacturers and users may not be experts in the field and must rely on certification by other authorities.

Although authentication is important, as this paper demonstrates, it seems as though manufacturers are not yet willing to sacrifice large portions of silicon to implement it. The most significant constraint, then, is to strike a balance between functionality and size.

4 Bitstream Authentication

Table 2. Parelkar's results targeting 90 nm ASIC architecture shown as the ratio of AES-128 in combination with SHA and AE schemes to stand-alone AES-128 [12]

criteria	AES-128(ECB)	+SHA1	+SHA512	+EAX	+CCM	+OCB
area	1 (41,250 μm^2)	2.8	4.4	1.5	1.4	1.5
throughput	1 (1097 Mbit/s)	1	1.2	0.4	0.4	0.8

Many protocols exist that provide authenticity. This section evaluates the most often used protocols for their suitability under our constraints. To help with the analysis, Table 2 summarizes Parelkar's results [12] that provides an estimates of the size and performance of the circuits in 90 nm technology.

4.1 Public-key Cryptography and Keyed-Hashes

PKC is theoretically suitable for bitstream encryption but is costly in hardware. The large keys involved and the many exponentiations and additions on wide bus length [13] make it unattractive for our application as it does not meet our constraint of small area. As mentioned, PKC is also slow compared to symmetric ciphers, and is mostly used for signatures and establishing session keys. As we shall see, there are faster and more nimble algorithms available for achieving our goal.

Keyed-hash message authentication codes (HMAC) produce a fixed length MAC from an arbitrary length message. The addition of the key into the input of a hash function provides authenticity [14]. A commonly used HMAC function is NIST's *Secure Hash Algorithm* (SHA) with various output lengths. Recently, flaws have been found in SHA-1 [15] and while these flaws are only of theoretical threat, it would be prudent to consider the stronger successor, SHA-2 [16] or its widely used variants, SHA-256 and SHA-512. If we use an AES-128 encryption core as the base for comparison, Parelkar's results as shown in Table 2, indicate that SHA-512 in combination with AES is 4.4 times as large as stand-alone AES, making this option unattractive. Other solutions may prove more efficient.

4.2 Authenticated Encryption

Table 3. Properties of authenticated encryption algorithms

protocol	passes	provably secure	associated data	free	standard
IAPM,XECB,OCB	1	yes	no	no	no
EAX, CWC	2	yes	yes	yes	no
CCM	2	yes	yes	yes	NIST

Since the late 1990s, there has been increased interest in protocols that securely provide encryption and authentication using a single key; there are collectively called *authenticated encryption* (AE) protocols. Black provides a concise introduction to AE algorithms in [17] and a summary of their properties is shown in Table 3. "Single- or dual-pass" AE, as the name suggests, refers to the times the message needs to be processed. Dual-pass algorithm perform each operation in a separate pass. "Provably secure" means that the scheme has been shown to be at least as secure as the underlying symmetric-key cipher it is based on. "Associated data" is a portion of the message that needs to be authenticated but not encrypted. An example is the routing header of a network packet. Since only the source and destination know the key, this header can not be encrypted and the destination to be sure that the packet has indeed arrived from the right source, it must be authenticated. Complete operational description of each of the

AE schemes is outside the scope of this paper and only the relevant attributes will be discussed.

Parelkar was the first to suggest the use of AE for FPGA bitstreams and concluded in [18,12] that the dual-pass *Counter with CBC-MAC* (CCM) would be best suited for bitstream authentication with the added benefit of it being a NIST recommended mode of operation [19]. Parelkar and Gaj also suggested EAX for bitstream processing in [20].

Dual-pass AE are not well suited for bitstream processing as they would require significant changes, circuit additions, and increased time to the configuration process. The reason is the lack of accessible temporary storage for the bitstream within the FPGA. After the first pass, the data has been loaded onto the configuration cells and if a second pass is needed, this data would need to be read-back, passed through the internal configuration processor again, and rewritten. In addition, not all of the bitstream's content, such as "no-ops", header and footer commands are stored in the configuration memory, making a second pass even more problematic. This complication can not be solved by sending the bitstream twice since, as stated, the security boundary is set at the FPGA, so there is no way to guarantee that the same bitstream will be received on both passes, allowing the attacker sending modified bitstreams for each pass, compromising the security.

Single pass algorithms are better suited, but currently, they are all proprietary. This is not a major hurdle for the FPGA manufacturers, but is still a hindrance for adoption given the other, royalty-free, options. Manufacturers would rather stay out of constrictive licenses for their hardware because in case of dispute, they can not simply "patch" the device. The other issue of lack of confidence in non-"approved" protocols has been previously mentioned.

The main disadvantage of AE for bitstream processing, however, is in what it was originally trying so solve: the separation of authentication and encryption. As we have seen in the examples of Section 2, there are applications that actually benefit from authentication-only schemes.

AE schemes do not seem to be ideal. Dual-pass schemes will require significant overhead while single-pass ones require licensing. Some modes, such as *Offset Codebook* (OCB), require that both the encrypt and decrypt functions be implemented at the receiving end, the FPGA. In addition to that, designers are averse to using new schemes that are not standardized or widely used; many AE protocols are still being evaluated. Having said that, in the future, AE schemes may become more appealing and their attributes better suited for our application.

4.3 Proposed Scheme

When discrete operations are combined to perform encryption and authentication the result is called *generic composition* and can be applied in the following ways: *MAC-then-encrypt*, *encrypt-then-MAC*, or *MAC-and-encrypt*. The benefits and disadvantages of each are discussed in [17] and [2, pp. 115–117] and needs to be chosen according to the application. The following proposed scheme is a

generic composition where two symmetric key ciphers operate in parallel on the same bitstream, one for decryption and the other for producing a MAC.

The CBC mode was briefly described in Section 2.1 when it is used for encryption, but it can also be used to produce a MAC. The last encrypted block of CBC is a MAC of the message since it depends on all previous blocks. CBC, however, has security shortcomings for variable length messages which may be an issue with partial reconfiguration and the variation of the bitstream's length, such as adding no-ops or commands after the device has been released. These deficiencies are corrected in *Cipher-based MAC* (CMAC) mode, which was recommended by NIST in 2005 [21]. CMAC is a wrapper around the symmetric block cipher and only uses the encryption/forward function.

The straight forward generic composition would be to have the two ciphers operate in parallel, one in CBC mode for decryption, and the other in CMAC mode, for authentication. This would mean both a decryptor and encryptor need to be implemented. If the amount of resources is to be minimized, this would be a disadvantage. As mentioned, AES's most resource demanding element is the s-box. For AES, these s-boxes are inverse of each other for the encrypt and decrypt functions, and therefore, must be implemented, or generated, separately. More effectively, in terms of hardware utilization, would be to use another mode for decryption that only uses the forward function.

Counter (CTR), *output feedback* (OFB), and *cipher feedback* (CFB) are all modes that use the forward function for both encryption and decryption. Using these modes allows the generic composition of two encryption cores to share resources for a more efficient implementation. Since the s-boxes are the same, their look-up tables can be shared. This may be done by multiplexing them within a clock cycle, as they can be made to be fast compared to other operations.

Readback, available in some FPGAs, allow the user to retrieve the current state of the FPGA while it is still in operation. Currently, when bitstream encryption is used, readback is disabled since the configuration data would come out in plaintext, defeating the purpose. The use of an forward-only mode will enable the data to be encrypted as it is read out of the FPGA.

The use of the same key for both operations may seem appealing since it will require only a single key-scheduling circuit. In general, this is bad practice that often leads to vulnerabilities and should be avoided. Aside from not having a proof of security, using the same keys would not enable the complete separation of operations. In some applications there may be a need for different distribution of keys where, for example, many devices share the same authentication key while having different encryption keys.

The proposed scheme is depicted in Figure 1 and operates as follows. The construction of the bitstream is very similar to the one used today with the exception of replacing the linear checksum (such as CRC) with a MAC. In the cases where authentication is not used, the CMAC can replace the checksum for all configurations by using a key of value 0. As an option, the implementation may allow the user to choose whether to MAC-then-encrypt or encrypt-then-MAC according to their preference, with the former resulting in a latency of a

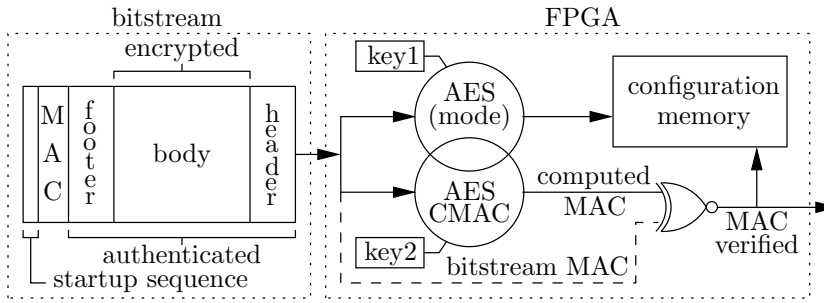

Fig. 1. Two parallel AES cores provide decryption and authenticity. The amount of shared resources depends on the modes

single block. The associated data, namely, the header and footer that provide the FPGA with initialization information and instructions is authenticated but not encrypted, preventing an attacker from enabling or disabling certain commands. The only portion of the bitstream that will not be authenticated is the startup sequence that could be removed if the commands are hard coded into the FPGA. The encrypted portion is the design itself that needs to be kept secret. After the footer has been processed, the computed and bitstream's MACs are compared. If they are equal, the FPGA will continue to the startup sequence. Otherwise, configuration will abort and the cells be cleared. After configuration has succeeded, the CMAC core can continue to operate without interfering with the configuration content or the operation of the device. This can be used for the authenticated heart beat described in section 2.4 or for authenticating partial reconfigurations.

5 Conclusion and Future Work

Encryption protects the design but does not prevent malicious code from running on the FPGA. Authentication allows the FPGA to operate as intended and be configured only from a bitstream generated by someone with whom it shares a secret. The examples given show that many applications can benefit from authentication. With the purpose of arriving at an appealing scheme that may be adopted by manufacturers, the constraints of the configuration environment were considered to narrow down the various protocols that provide authentication. The solution proposed allows for the separation to the two processes and does not demand significant increase to resources beyond what is currently used for bitstream encryption. The solution also has minimal impact on the current configuration process.

Future work includes ASIC implementations to quantify resource-sharing using the various modes of operation and optimization techniques.

Acknowledgements

Thanks to Steven J. Murdoch, Markus Kuhn, Ross Anderson, Milind Parelkar, and anonymous reviewers for their valuable comments and suggestions.

References

1. NIST, U.S. Dept. of Commerce: FIPS 197: Advanced encryption standard. (2001)
2. Ferguson, N., Schneier, B.: Practical Cryptography. John Wiley & Sons, Inc., New York, NY, USA (2003)
3. Dworkin, M.: Special Publication 800-38A: Recommendation for block cipher modes of operation. NIST, U.S. Dept. of Commerce. (2001)
4. Hadžić, I., Udani, S., Smith, J.M.: FPGA viruses. In: FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications. Volume 1673 of LNCS., London, UK, Springer-Verlag (1999) 291–300
5. Stigge, M., Plötz, H., Müller, W., Redlich, J.P.: Reversing CRC – theory and practice. Technical Report SAR-PR-2006-05, Humboldt University Berlin (2006)
6. Baetoniu, C., Sheth, S.: XAPP780: FPGA IFF copy protection using Dallas Semiconductor/Maxim DS2432 Secure EEPROM. Xilinx Inc. (2005)
7. Altera Corp.: FPGA design security solution using MAX II devices. (2004)
8. Xilinx Inc.: UG332: Spartan-3 generation configuration user guide. (2006)
9. Xilinx Inc.: DS202: Virtex-5 data sheet: DC and switching characteristics. (2006)
10. Altera Corp.: Stratix III design handbook. (2006)
11. Xilinx Inc.: DS099: Spartan-3 FPGA family: Complete data sheet. (2006)
12. Parelkar, M.M.: Authenticated encryption in hardware. Master's thesis, George Mason University, Fairfax, VA, USA (2005)
13. Batina, L., Örs, S.B., Preneel, B., Vandewalle, J.: Hardware architectures for public key cryptography. VLSI Journal, Integration **34**(1-2) (2003) 1–64
14. NIST, U.S. Dept. of Commerce: FIPS 198: The keyed-hash message authentication code (HMAC). (2002)
15. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: CRYPTO. (2005) 17–36
16. NIST, U.S. Department of Commerce: FIPS 180-2: Secure hash standard. (2002)
17. Black, J.: A. Authenticated encryption. In: Encyclopedia of Cryptography and Security. Springer (2005) 10–21
18. Parelkar, M.M.: FPGA security – bitstream authentication. Technical report, George Mason University (2004) `http://mason.gmu.edu/~mparelka/reports/bitstream_auth.pdf`.
19. Dworkin, M.: Special Publication 800-38C: Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality. NIST, U.S. Dept. of Commerce. (2005)
20. Parelkar, M.M., Gaj, K.: Implementation of EAX mode of operation for FPGA bitstream encryption and authentication. In: Field Programmable Technology. (2005) 335–336
21. Dworkin, M.: Special Publication 800-38B: Recommendation for block cipher modes of operation: The CMAC mode for authentication. NIST, U.S. Dept. of Commerce. (2005)