Computer-assisted machine-to-human protocols for authentication of a RAM-based embedded system

Abdourhamane Idrissa, Alain Aubert, Thierry Fournel

Computer-assisted machine-to-human protocols for authentication of a RAM-based embedded system

Abdourhamane Idrissa, Alain Aubert and Thierry Fournel

Laboratoire Hubert Curien,CNRS UMR5516
Universite de Saint-Etienne, F-42000, Saint-Etienne, France
Universite de Lyon, F-69003, Lyon, France

ABSTRACT

Mobile readers used for optical identification of manufactured products can be tampered in different ways: with hardware Trojan or by powering up with fake configuration data. How a human verifier can authenticate the reader to be handled for goods verification ?

In this paper, two cryptographic protocols are proposed to achieve the verification of a RAM-based system through a trusted auxiliary machine. Such a system is assumed to be composed of a RAM memory and a secure block (in practice a FPGA or a configurable microcontroller). The system is connected to an input/output interface and contains a Non Volatile Memory where the configuration data are stored. Here, except the secure block, all the blocks are exposed to attacks.

At the registration stage of the first protocol, the MAC of both the secret and the configuration data, denoted M_0 is computed by the mobile device without saving it then transmitted to the user in a secure environment. At the verification stage, the reader which is challenged with nonces sends MACs / HMACs of both nonces and MAC M_0 (to be recomputed), keyed with the secret. These responses are verified by the user through a trusted auxiliary MAC computer unit. Here the verifier does not need to tract a (long) list of challenge / response pairs. This makes the protocol tractable for a human verifier as its participation in the authentication process is increased. In counterpart the secret has to be shared with the auxiliary unit. This constraint is relaxed in a second protocol directly derived from Fiat-Shamir's scheme.

Keywords: machine-to-human authentication, machine integrity, challenge-response protocol

1. INTRODUCTION

Nowadays various tasks are automatically ensured off-line once the dedicated machine is verified by the user. A machine-to-Human authentication scheme is achieved by involving the user active participation in order to be convinced of the process integrity. In goods authentication, the proper functioning of the optical reader is verified at the opening session by an authorized person. Even if the operating module is a secure one, the input/output channel is usually still vulnerable to potential attacks. An attacker could intercept and manipulate the digital image on the way to the secure module then undo the modifications when the module sends it back. An attacker can also replace the electronic core of the optical reader by a malicious system able to simulate the behavior of the authentic one.

As in challenged Human-to-machine authentication, verification based on cryptographic key ([1–3]) or zero-knowledge protocol[4] can then be applied. However in these processes, the human user is passive. The question is how the verifier can be involved in the off-line procedure for checking the authenticity of the hardware supporting the embedded system RAM and the content to be downloaded in it. This problem concerns numerous embedded systems as systems based on Field-Programmable Gate Array (FPGA) or microcontroller. In some FPGAs, the SRAM is configured via an internal controller according to a bitstream initially stored in an external Non Volatile Memory (NVM). In micro-processors, a RAM is set up by a controller unit with instructions and data which are stored in an external NVM.

In this paper we suggest an interactive off-line authentication of the embedded system (the prover) by the user (the verifier) who is helped with a trusted computation tool. The general architecture of RAM-based embedded systems and associated threats are reported in section 2. In section 3 two authentication protocols are described, security and reliability are discussed.

Further author information: (Send correspondence to Thierry Fournel)
E-mail: fournel@univ-st-etienne.fr, Telephone: +33 (0)4 77 91 57 80

2. RAM DATA AUTHENTICATION AND THREATS

An embedded system is classically built from a microcontroller or/and a FPGA. A microcontroller executes an application code which is stored in its embedded RAM memory. A FPGA is architectured according to a binary file called a bitstream stored in its embedded RAM memory. In these cases, RAM is loaded on power-up by the content of a non volatile memory (NVM) which contains instructions and data (forming the application code) for a microcontroller, the image of the FPGA underlying architecture (the bitstream) for a FPGA. Therefore these data define the embedded system and their authentication is crucial.

The main protection consists in computing a Message Authentication Code (MAC) of the data to be loaded in RAM memory thanks to a secret key $Kmac$, in a secure place. The MAC is then stored in the NVM together with the genuine data. Later the loading controller will compute a MAC of the current data in the NVM and compare the computed MAC to the stored one. If the MACs are different, the loading controller will not load data into the RAM memory. In FPGAs, the loading controller corresponds to a specific secure area containing a MAC computation function plus a dedicated secret key. It allows the data loading from the external NVM (to the embedded RAM): the User Logic (Figure 1 and 2). In microcontrollers, the program bootloader can play the role of the loading controller by computing MACs and downloading from the NVM.

Concerning FPGAs, the specific area is called Configuration Logic in[5] or Secure update mechanism in.[6] The latest Actel devices[2] and series 6 Xilinx FPGAs (Spartan6 and Virtex6)[1] include cryptographically secure mechanisms to ensure integrity. Actel implements an AES block for integrity checking (AES-based MAC): a MAC is included in the bitstream and the configuration (or loading) controller logic checks it before starting the design. Series 6 of Xilinx FPGAs include an integrity checking engine in the static logic. For example, Virtex-6[1] devices have an on-chip bistream keyed-Hash Message Authentication Code (HMAC) using algorithm SHA-1. During the configuration process, the HMAC/SHA256 engine computes the MAC and compares it with the MAC in the bitstream. If the two MACs match, the configuration goes to completion through the startup cycle. If the two MACs do not match, the controller configuration logic disables the configuration interface, blocking any access to the FPGA.

Concerning microcontrollers, in Maxim Zatara ZA9L1,[3] secure 32-Bit ARM microcontrollers, the authenticity of the application code is first cryptographically verified by applying SHA256 on power up to ensure that attackers cannot insert their own application code.

However, the mechanisms mentioned above are not resistant to the replay of an old version of the data having security faults. It is the reason why the identification number of the version has to be taken into account. Badrigans et al have proposed in[6] a FPGA architecture which prevents replay attacks.

In all the embedded systems previously described, the human verifier remains passive and relies on the self-authentication of the system on power-up. Unfortunately a user without external communication is not protected against a *block substitution* i.e. replacing the secure block with another one and simultaneously changing the content of the NVM to control the embedded system. Without an external communication with a trusted authority or a physical inspection of the embedded system, a human verifier cannot detect a block substitution. We suggest to detect such a tampering and authenticate embedded systems by making the verifier an active user through a challenge/response protocol then through an adaptated Fiat-Shamir's zero-knowledge protocol.

3. AUTHENTICATION PROTOCOL BY A HUMAN VERIFIER

Let us now consider the architecture shown in Figure 1. It is formed with a circuit composed of a secure block defined by the system configurable RAM and the associated controller. The modules that do not contribute to the configuration of embedded systems are not represented. The secure block can be a FPGA as shown in Figure 2 or a configurable microcontroller. In such an architecture, all the keys stored in the configuration controller together with the associated functions are securely protected. Besides, all the connections with the configuration controller are unsecure as the connections to the RAM, I/O (Input/Output interface) and NVM.

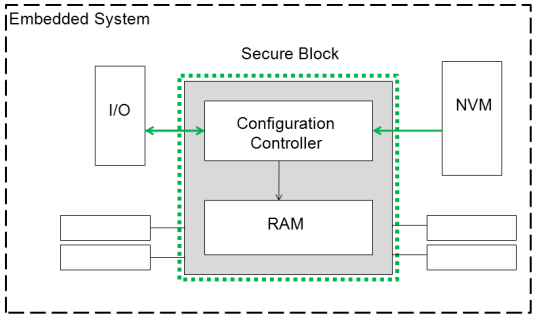

Figure 1. Architecture of a RAM-based embedded system

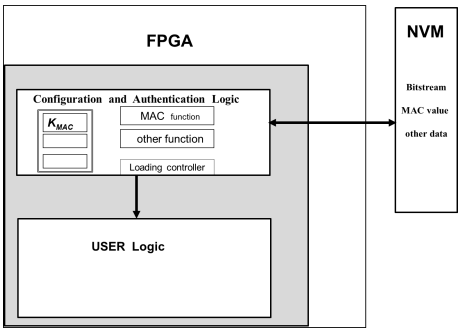

Figure 2. Configuration of a FPGA chip

3.1 A first challenge-response protocol

Here the embedded system is assumed to have an unique secret key, $Kmac$ in its configuration controller. We propose to use an additional secret denoted as S, also stored in the configuration controller. This secret is randomly chosen by the user at the registration phase to be achieved in a safe environment i.e. without any passive or active threat. During this phase, the MAC value of both the configuration data stored in the NVM and the secret, M_0 is computed by the configuration controller then forwarded to the user.

Value M_0 that is used as the fingerprint of the RAM content of the circuit has to be retrieved at the verification phase by requesting to the configuration controller the computation of the current value of the MAC.

In this way, the user can verify both the presence of the secret and the integrity of the RAM content by comparing the current MAC, M_0' to the original one, M_0. Such a protocol makes use of symmetric cryptographic functions already implemented in some embedded systems that seek to verify configuration contents - e.g. Actel Fusion[2] and Virtex6[1] - by computing a MAC value of the configuration data (with either a symmetric encryption function or a hash function).

However, MAC value M_0 must not be divulged to force its subsequent re-computation by the configuration controller and to avoid a replay attack. Challenges/responses can therefore be suggested to convince the verifier without divulgation. Challenges can be composed with nonces and responses with both nonces and MAC values. To be tractable by a human verifier (in a reasonable time), MAC verifications will be achieved for him by an external, trusted computer.

3.1.1 Registration phase

The enrolment phase is as follows (Figure 3):

1. *User:* For each registration, the user randomly selects a secret S having the same bit length as $Kmac$, and sends secret S to the configuration controller.
2. *Embedded System:* Upon receiving user's secret S, the embedded system stores it in its secured area (configuration controller), reads the configuration data in the NVM and computes MAC value ($M_0 = Mac_{Kmac}(S\|$"NVM data")) of both with its MAC key $Kmac$. Afterwards, the embedded system returns the MAC value M_0 to the user.

3.1.2 Verification Phase

Here we neglect the risk of denial of service i.e. any deliberate or accidental modification of data stored in the embedded system.

The human verifier who knows secret S is helped with an external computer where are embbeded secret S and an implementation of the same MAC function as that of the embbeded system. This external computer is assumed to be non-corrupted. In particular, it does not disclose any information about secret S.

Verification phase takes place as follows (schematized on Figure 3).

1. *User:* The user sends a random nonce N_{us} to the embedded system through the I/O interface.

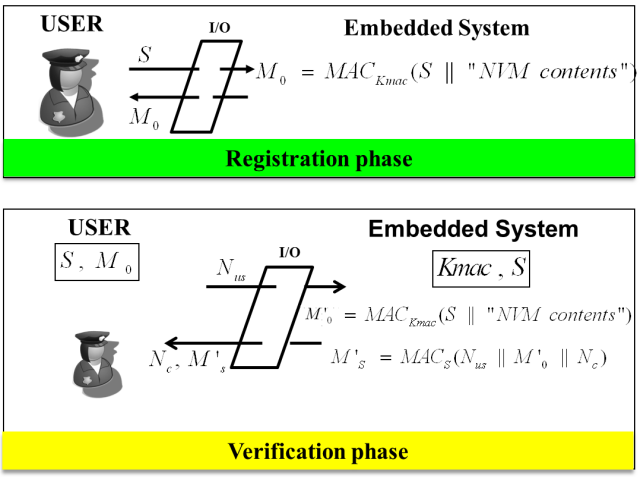

Figure 3. Authentication protocol based on MAC calculation

2. *Embedded system:* After reception, the configuration controller initiates the reading of the NVM and computes MAC $M_0^{'}$ then the MAC value of a new nonce N_c concatenated with $M_0^{'}$ and N_{us}, denoted as $M_S^{'}$ where secret S is used as MAC key. The embedded system returns nonce N_c and MAC value $M_S^{'}$ to the user.
3. *User:* The user inputs MAC value M_0, nonces N_{us}, N_c and secret S into the trusted tool.
4. *Trusted tool:* The MAC value of the inputs, M_S is computed by the trusted tool then displayed.
5. *User:* The user checks if MAC values M_S and $M_S^{'}$ are the same (”Yes”) or not (”No”).

The process is iterated a number of times such that the user can detect a tampering of the embedded system by checking that a ”No” is not so rare.

3.1.3 Discussion about security

The protocol described in this paper like the ones mentioned in literature[1,2,5,6] is based on the trust and security of the configuration controller. We assume that it is safe and always computes a correct value of MAC (without storing the MAC values in the circuit), i.e. $M_0^{'}$ corresponds to the MAC value obtained from the current NVM data. Nonce values N_{us} are used such that only the knowledge of S can allow the computation of MAC values $M_S^{'}$.

Between two changes of the secret by the user (by performing a new registration), secret S has to be input into the external tool (for MAC computation). Thus, there is a risk of disclosure of secret S. To avoid sharing user's secret with an external device, we propose below to modify accordingly Fiat-Shamir's zero-knowledge protocol.

3.2 Zero-Knowledge Authentication

The second is to adapt Fiat-Shamir's authentication protocol, by describing a zero-knowledge protocol between configuration controller and the human user. And in this case the zero-knowledge protocol allows the user not to share secrets with the external trusted calculation tool. The calculator will just be used to perform arithmetic operations needed to perform the verification.

3.2.1 Fiat-Shamir Authentication Protocol

In,[4] Fiat and Shamir propose an interactive zero-knowledge protocol based on the difficulty of extracting modular square roots when the factorization of n is unknown. Using this property, the protocol allows a *Verifier* V to check the authenticity of a *Prover* P, without the prover divulges a secret to the verifier. The assumptions and parameters of the authentication scheme is indicated below.

Parameters:

- $n = p \cdot q$, where p and q are secret prime number.
- A public integer k.
- Secret values $(s_1, \cdots, s_k)$, such that $s_i < n$, and $\gcd(s_i, n) = 1$ for all i.
- Public values $(y_1, \cdots, y_k)$ such that $y_i = x_i^2 \pmod{n}$

Assumptions:

- The values $n, (y_1, \cdots, y_k)$ are public, i.e. the verifier knows n and $(y_1, \cdots, y_k)$.
- The integers $(s_1, \cdots, s_k)$ are secret, i.e. only the prover P knows $(s_1, \cdots, s_k)$ values.

The protocol has the following items:

1. *Prover:* P generates a random integer r, with $r < n$, and sends to $V : c = r^2 \pmod{n}$.
2. *Verifier:* V sends to P a binary vector
$$B = (b_1, \cdots, b_k) \in \{0, 1\}^k.$$
3. *Prover:* The prover P sends to V :
$$u = \frac{r}{s_1^{b_1} \cdots s_k^{b_k}} \pmod{n}.$$
4. *Verifier:* V verifies that
$$c = u^2 \cdot y_1^{b_1} \cdot y_2^{b_2} \cdots y_k^{b_k} \pmod{n}.$$

The protocol is repeated with different values for r and B until the verifier is convinced that P knows the secret values $(s_1, \cdots, s_k)$.

The correctness and security of the protocol is proved in.[4]

We describe below how to adapt this zero-knowledge protocol to the machine-to-human authentication of a RAM-based embedded system.

In addition to the MAC function, the derived protocol will require arithmetic functions in the configuration controller. The value of module n must be defined and stored safely in the configuration area of the circuit, however it is not necessary for either the user or the circuit to know the factorization of n. Similarly parameter k,, the number of secrets to be calculated must be provided and fixed in advance.

3.2.2 Registration Phase

During the registration phase of the protocol described in the previous section 3.1 the user must provide his secret to the trusted tool (for MAC computation). Here, the idea is to define secret values from the NVM content and user's secret S. Only the public parameters are know by the user.

The registration phase is as follows:

1. *User:* With the same principle like the protocol described in section 3.1, for each registration process, the user picks a random secret S (here user's secret can be any bitstring) and sends S to the circuit.
2. *Embedded System:* Upon receiving user's secret S, the configuration controller reads the NVM content and computes s_i and y_i for all $i \in \{1, \cdots, k\}, y_i$:
$$s_i = Mac_{Kmac}(S \| \text{"NVM contents"} \| i), \quad (1)$$
$$y_i = s_i^2 \pmod{n}. \quad (2)$$

While $\gcd(s_i, n) \neq 1$, for some i, the process is restarted (the user picks a value for S). The system stores secret S in its secure area and returns public values $(y_1, \cdots, y_k)$ to the user.

Data S, n and $(y_1, \cdots, y_k)$ constitute the informations that will allow the verification phase of the protocol. The value of S must be kept secret until a new registration phase in order to authenticate the circuit and to prevent the replay of an old NVM content. The user must prevent the public $(y_1, \cdots, y_k)$ values from change.

3.2.3 Verification Phase

The verification consists in checking if the imprint of the contents of the NVM corresponds to that computed during the registration phase, $(y_1, \cdots, y_k)$ (see Equation (1)). Here, the *prover* P is the configuration controller of the embedded system and the *verifier* V is the human user. So the user needs an trusted external tool to calculate modular multiplications. This trusted tool can be any reliable system to calculate the multiplication modulo n. In contrary to the verification of above protocol based on the MAC calculation, no secret informations are shared with the calculator. Verification takes place as follows :

1. *Embbeded system:* The authentication process starts at RAM configuration, The configuration controller reads the configuration from the NVM then regenerates the secret $(s_1, \cdots, s_k)$ with secret S and key $Kmac$
2. *User* $\leftrightarrow$ *Embbeded system:* With the following exchanges, the user aided by a Trusted tool , checks the computed values of $(s_1, \cdots, s_k)$ relative to n and $(y_1, \cdots, y_k)$.
   (a) *Embbeded system:* The configuration controller generates a random integer r, with $r < n$, and sends to the User $c = r^2 \pmod n$.
   (b) *User:* The user choses and sends to Embbeded system a binary vector $B = (b_1, \cdots, b_k) \in \{0, 1\}^k$.
   (c) *Embbeded system:* computes and returns to user $u = \frac{r}{s_1^{b_1} \cdots s_k^{b_k}} \pmod n$.
   (d) *User:* The user inputs value u, the binary vector B, the module n and the check values $(y_1, \cdots, y_k)$ into the trusted tool.
   (e) *Trusted tool:* With the inputs data, a number c' is computed by the trusted tool then displayed to the user.

$$c' = u^2 \cdot y_1^{b_1} \cdot y_2^{b_2} \cdots y_k^{b_k} \pmod n.$$

   (f) *User:* The user checks if MAC values c and c' are the same ("Yes") or not ("No").

   The process (2a to 2f) is iterated a number of times such that the user can detect a tampering of the embedded system by checking that a "No" is not so rare.

4. CONCLUSION

In this paper we suggest two protocols for an off-line machine-to-human authentication of RAM-based embedded systems. The user is active even if he is helped with a trusted computing tool. This tool allows MAC computations in the first protocol, modular multiplications in the second one. The last protocol avoids to share a secret with the tool thanks to a Fiat-Shamir zero-knowledge scheme. We are working to suppress the trusted computer in order to get an even more active user participation. Such a scheme will have to be adapted to the human capabilities and offer a certain resistance to machine cryptanalysis.

ACKNOWLEDGMENT

The present work is funded by Region Rhône-Alpes (France) in the frame of Cluster ISLE (Informatique, Signal et Logiciel Embarqué). The authors would like to thank Prof. Roland Gillard but also Signoptic Technologies for their constant encouragement.

REFERENCES

[1] Xilinx Inc., *UG360: Virtex-6 FPGA Configuration user guide* (November 2010).

[2] Actel Corp., "Fusion and extended temperature fusion fpga fabric user's guide," (July 2010).

[3] Maxim Integrated Products, Inc, *ZA9L1: Zatara High-Performance, Secure, 32-Bit ARM Microcontroller*. maxim-ic (march 2009).

[4] Fiat, A. and Shamir, A., "How to prove yourself: Practical solutions to identification and signature problems," in [*Advances in Cryptology - CRYPTO'86*], Odlyzko, A., ed., *Lecture Notes in Computer Science* **263**, 186–194, Springer Berlin / Heidelberg (1987).

[5] Drimer, S. and Kuhn, M., "A protocol for secure remote updates of fpga configurations," in [*Reconfigurable Computing: Architectures, Tools and Applications*], Becker, J., Woods, R., Athanas, P., and Morgan, F., eds., *Lecture Notes in Computer Science* **5453**, 50–61, Springer Berlin / Heidelberg (2009).

[6] Benoît, B., Reouven, E., and Lionel, T., "Secure fpga configuration architecture preventing system downgrade," in [*Field-Programmable Logic and Applications*], 317–322 (2008).