Secure JTAG implementation using Schnorr Protocol

Amitabh Das[1], Jean Da Rolt[2], Santosh Ghosh[1], Stefaan Seys[1], Sophie Dupuis[2], Giorgio Di Natale[2], Marie-Lise Flottes[2], Bruno Rouzeyre[2], and Ingrid Verbauwhede[1]

[1] KU Leuven and IMinds, ESAT/COSIC, Leuven, Belgium
{firstname.lastname}@esat.kuleuven.be

[2] LIRMM (Université Montpellier II /CNRS UMR 5506), Montpellier, France
{darolt, dinatale, flottes, rouzeyre, dupuis}@lirmm.fr

Abstract. The standard IEEE 1149.1 (Test Access Port and Boundary-Scan Architecture, also known as JTAG port) provides a useful interface for embedded systems development, debug, and test. In an 1149.1-compatible integrated circuit, the JTAG port allows the circuit to be easily accessed from the external world, and even to control and observe the internal scan chains of the circuit. However, the JTAG port can be also exploited by attackers to mount several cryptographic attacks. In this paper we propose a novel architecture that implements a secure JTAG interface. Our JTAG scheme allows for mutual authentication between the device and the tester. In contrast to previous work, our scheme uses provably secure asymmetric-key based authentication and verification protocols. The complete scheme is implemented in hardware and integrated with the standard JTAG interface. Detailed area and timing results are also presented.

Keywords: JTAG, secure testing, IP protection, secure code and firmware updates, cryptographic circuits, Schnorr protocol, Elliptic Curve Cryptography, mutual authentication.

1. Introduction

Joint Test Action Group (JTAG) is the common name for what was later standardized as the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture [1]. JTAG has remained as the ubiquitous test and debug interface standard for circuits and printed circuit boards in the semiconductor industry for more than two decades. The companion standard, IEEE Standard 1532 (Boundary-Scan-Based In-System Configuration of Programmable Devices) has extended JTAG to support on-board programming [2]. A current IEEE standard proposal (P1687, also known as Internal JTAG) seeks to further enhance JTAG by allowing block transfer of data and special instruction sets in order to speed up In-System Programmability.

JTAG was initially designed without a concern for security. As the capability of hardware attackers is increasing, more and more side-channels are discovered, which can compromise the security of a device. One such important side-channel is the improper use of the JTAG port. There have been many practical attacks on secure devices such as set-top box (STB) decoders using the JTAG interface [3]. ARM11 (Cortex) microcontroller, which is used in latest smartphones, has extensive test and debug facilities through the JTAG port. This is a well-known backdoor that is currently used for instance to jailbreak iPhones/iPad, or to unlock protected services in mobile phones [4]. Even if not documented, it is reasonable to think that JTAG could be used to compromise the security of other applications such as mobile e-payments, or Wireless Sensor Nodes (WSNs) [5, 6].

Another security flaw due to JTAG is related to FPGAs. The configuration bitstream which contains the Intellectual Property (IP) information of a reconfigurable design is mostly programmed via the JTAG interface into FPGAs [7]. The firmware update of set-top boxes used in pay-TV subscriptions also happens in most cases through the JTAG port. An insecure JTAG access would allow on one side to re-program parts of the system at the hacker's will, and on the other side it could be used to sniff configuration bits thus allowing retrieving the IP information.

Though there are several approaches for securing the JTAG interface, which can be found in the literature [8, 9, 10, 11, 12], most of them are based on symmetric-key approaches. They have an inherent key management problem. This is what we intend to overcome through the use of Public-key Cryptography (PKC) in our secure JTAG scheme. Though there is previous work on a protected JTAG scheme using ECC-based authentication protocol [26], the scheme uses PKC in a non-standard way causing key-management problems. Moreover, the paper also does not present any timing or area results. Though several PKC protocols can be used for establishing a secure authentication for JTAG, we use the ECC-based Schnorr protocol which is an efficient and provably secure protocol [13].

In this paper, we seek to provide security features to the IEEE 1149.1 JTAG interface by including a Schnorr-based secure test protocol, and present an efficient hardware implementation of the protocol using elliptic curve cryptography. Moreover, our approach does not make any modifications to the existing JTAG interface. To the best of our knowledge, this is the first paper that proposes a mechanism for mutual authentication between the secure device and the tester based on a well known and studied public key authentication protocol. Earlier work is either based on symmetric key systems or only proposes one way authentication, limiting the scenarios in which these systems can be used. The area requirement to incorporate this secure test infrastructure on the JTAG has been optimized to increase the scope of our proposed scheme in a wide range of application scenarios.

The rest of the paper is structured as follows. In Section 2, we present the past work that has been done in the area of secure JTAG implementation. A comparison of these approaches with our scheme is also made. The motivation for our work is given in Section 3. Section 4 presents the attacker model for the JTAG mechanism, and the idea for the Schnorr-based authentication protocol that is the basis of our secure JTAG strategy. This section also includes a discussion on the public key authenticity. Section 5 presents the Secure JTAG implementation. Two designs are presented using affine and projective coordinates. The area and timing results are shown in Section 6 and we conclude the paper in Section 7.

2. Previous Work

An ordinary JTAG standard [1] consists of a pre-defined interface, containing a serial input called TDI, a serial output called TDO, an input for the clock TCK and a mode select input called TMS. By controlling the TMS signal, the user can travel between the 16 states of the JTAG finite state machine, shown in Appendix F. Then, the request for executing the instructions and the transference of data between the circuit and the host is performed by connecting the input TDI and the output TDO to internal shift registers. Thus a malicious host can manipulate the JTAG inputs and execute any instruction.

One of the first approaches for implementing a secure JTAG appears in [8]. It presents a locking/unlocking mechanism for controlling the access to the JTAG instructions. It is based on storing a secret key inside the chip boundaries. To gain access to the JTAG features the user must shift in the secret key, otherwise the JTAG bypasses all the data on the TDI input to the TDO output. The scope of this approach does not consider the case where a fake circuit requests updates which may compromise the intellectual property.

A detailed evaluation of the JTAG test standard, its security problems, attackers' capabilities, possible attacks and countermeasures has been done in [9]. It presents a JTAG security protocol using a stream cipher (Trivium), hash function and a message authentication code. The authors suppose that the service server is trusted, performing one-way authentication. However, to protect the data from unauthorized servers, the data is encrypted.

An anti-tamper JTAG Test Access Port (TAP) is described in [10] that uses SHA-256 secure hash and a true random number generator (TRNG) to create a low gate overhead challenge/response based access system employing an on-chip internal JTAG P1687 instrument. It is mentioned by the authors that malicious designers could modify the designs in order to observe the secret key, implying a one-way authentication scenario.

A multi-level security access system for controlling access to individual scan cells for preventing malicious opcodes from being loaded into the JTAG controller is presented in [11]. This approach also supposes the design is trusted, and thus it is not possible for fake circuits to obtain proprietary updates.

An elaborate three-party secure JTAG protocol using certificates involving SHA-1 hash algorithm, AES block cipher and several arithmetic operators is presented in [12]. The authors describe the possible attack cases, but the protocol is not proven to be secure.

There are also industrial solutions for providing security to the JTAG interface. The Secure JTAG Controller (SJC) which features in Freescale Semiconductors i.MX31 and i.MX31L Multimedia Applications Processors is one such example. Similarly there are tools available from various vendors such as Discretix and Lauterbach TRACE32 PowerTools, which provides Secure JTAG Debug module giving OEMs a highly secure, authenticated way to debug SoC errors throughout a system's lifetime. A detailed overview of the JTAG related fuses and security features in the AVR microcontroller can be found in [14]. Some use-cases and application scenarios involving JTAG security are presented in [15].

To the best of our knowledge, the work in [26] is the only JTAG security solution that is also based on asymmetric key cryptography. In contrast to our work, this solution only provides one-way authentication from the test server to the JTAG device. Moreover, this solution does not improve key management related to symmetric solutions, as it requires the test server to have secure access to a database that contains all the unique private keys related to each device. Because of the non-standard setup of the authentication protocol, every JTAG device has a unique private key that is stored in a database. This key is retrieved by the test server in order to authenticate itself to the device. In our solution, we employ a standard use of public/private keys in which the prover uses its own private key and a certificate signed by a CA to prove its authenticity and not a private key related to the verifier as in [26].

Most of the previous approaches [8, 9, 10, 11, 26] suppose a one-way authentication, where either the circuit or the server is considered trusted. In this paper, we propose a suitable solution in cases where neither the circuit nor the server is trusted. Additionally, most of the previous solutions are not provably secure as the Schnorr protocol used in this paper.

3. Motivation

JTAG is mainly used for manufacturing and in-the-field test-and-diagnosis of VLSI circuits and boards. It may be disabled in the chip-die after initialization of a product. However, there are some applications where JTAG is kept enabled for code or firmware updates. Especially, in case of reconfigurable devices like set-top boxes, where even remote reprogramming may be done through JTAG port based on updates received from the service provider. For instance, the STi7101 low-cost HDTV set-top box decoder and the TI MSP430 used in some set-top boxes have the JTAG open for product support and service.

In this work, we solve the inherent key-management problem of existing Symmetric-Key Cryptography (SKC) based secure JTAG approaches using Public-Key Cryptography (PKC). Specifically, if SKC is used for securing JTAG, there will be a common master secret key for all products or a large secret-key database needs to be maintained at the tester/updater side, which are not good options for mass electronic products. PKC implementations are inherently more hardware expensive and slower than SKC based approaches. Therefore it is a challenging task to incorporate PKC in a resource constrained environment like JTAG.

The use of asymmetric primitives and the related public/private key pairs substantially improve the complexity involved in key management in this setting of tester against the device. If we take for example the automobile industry, then we expect to bring our car to virtually any garage in the world and get our car serviced. Servicing cars now also includes updating software in one of the on-board units (OBUs) which may be through the JTAG interface. Currently these updates can be pushed to the OBU as soon as it is powered on; no other security measures are used. One of most important reasons for the current lack of authentication is the fact that it presents car manufacturers with a large key management problem that is inherent to the use of symmetric solutions in large scale systems. In symmetric solutions, the verifier needs a copy of the same key that was also used to generate the authentication token (e.g., a message authentication code or MAC on the firmware). This implies that the use of a single master key is very risky as it will be wide spread in many devices and likely to leak at one point in time (see Section 4 for more details on who is the verifier in our approach). Therefore, symmetric key based solutions require unique keys to be installed at every verifier. In large scale systems, this would require a large database that link the identity of the prover to its key and a means for verifiers to securely access and authenticate this service. Alternatively, key derivation schemes could be used, but they only lower the risk related to a single master key. There are many other application scenarios where similar key-management problems can occur.

To overcome this problem, the solution proposed in this paper offers the possibility of using certificates instead of shared symmetric keys. This would for example allow the use of the same signed firmware update for a wide range of OBUs, without the risk of installing the same symmetric key in this range of devices. They just need a valid copy of the manufacturers' public key for signature verification.

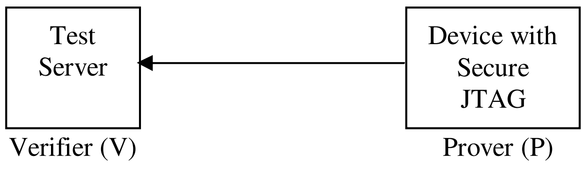
4. Proposed Secure JTAG Scheme

4.1 Attacker Model

We have considered the following application scenarios for our attacker model. The JTAG interface of a VLSI circuit is normally used for testing the device, as well as for updating the internal code and firmware in some applications. We assume that the external JTAG interface of the target device is enabled and is accessible to the attacker. In [9], the attacker models are described based on malicious IP cores inside a SoC. However, in this paper, we have considered the following two attacker scenarios where internal IP cores are assumed to be trusted, and the attacker is an external entity to the cryptographic SoC. We assume that it is impossible to extract the stored private key (present in secure memory) on the device containing the JTAG interface.

Manufacturing test/firmware or code update at manufacturer's end: We have considered the manufacturing environment to be controlled and the manufacturer's test server to be trusted. The device may be a fake one (or a clone) trying to get unauthorized code or firmware updates through the JTAG interface. The test server should allow only genuine devices to have access to the updates. Hence, in this scenario, a one-way entity authentication of the device to the Test Server is required.

The device needs to prove to the Test Server that it is in possession of the correct private key, without revealing it to the server. This is achieved through the use of the Schnorr protocol to be employed in our secure test scheme. Here the prover is the device with secure JTAG, while the verifier is the Test Server. This is represented symbolically by the block diagram below:

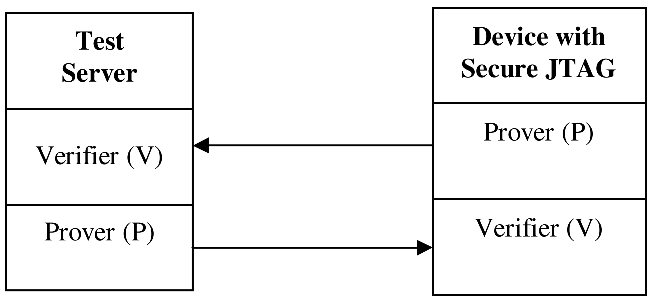


A possible use-case for this scenario is system integration, where the integrator procures VLSI chips from different third-party vendors. He needs to make sure that each chip is a genuine one, and not a fake one or clone which can compromise the integrity of the complete system. This can be achieved through the addition of a security feature to the JTAG using an authentication mechanism.

In-the-field update, debug and test: When devices are deployed in-the-field, the environment is considered uncontrolled and both the Test Server and the device with the JTAG may be potential attackers. Hence, mutual entity authentication is required between the device and the Test Server. The Test Server might be a hacker or a malicious user trying to extract the internal secrets from the device through the test infrastructure. Similarly, the device may be malicious or even a fake one, trying to procure unauthorized code or firmware updates through the JTAG interface.

Hence, both the device and the Test Server need to prove their identity to each other without revealing their secrets (their private keys). For the mutual authentication using ECC based Schnorr protocol, when the Secure JTAG is the prover, the Test Server is the verifier. Similarly when the Test Server is the prover, the Secure JTAG is the verifier.

This is represented graphically by the following block diagram:



A possible application of this attacker model is the firmware update of set-top boxes used in pay-TV subscriptions. The user of the set-top box might be a possible attacker trying to get an unauthorized update from the server using the JTAG port to watch pay channels for free. Similarly, an unauthorized update from a remote hacker using the JTAG port might compromise the secret keys stored in the smart card of the set-top box.

The Schnorr protocol itself is proven secure in a very strong attacker model [13]. This means that no information about the private key is leaked to the verifier or any of the attackers that fit the attacker model in [13]. As a result, the system can only be attacked by extracting the private key through side-channel attacks (though our designs are protected against Simple Power Analysis), leaks during installation/generation of the keys, attacks on the CA facilities, etc. In this paper we present an efficient implementation of the Schnorr protocol that makes its use cost effective for low cost JTAG devices. Side channel attacks on this implementation or attacks related to software bugs, etc. or not in scope of this paper. We claim that our solution is secure in the two scenarios described above, as it is a straightforward use of the Schnorr protocol that is proven secure.

4.2 Secure Test Authentication Based on Schnorr Protocol

We use an enhanced version of ECC-based Schnorr Protocol [13] as the public-key cryptographic protocol in our secure JTAG test scheme. Various public-key implementations, such as RSA or ECC, may be used to solve the key-management problems present in previous secure JTAG approaches. We chose ECC as it offers the same security as RSA, with much smaller area footprint. Area overhead is of critical importance, since we are constrained in terms of silicon area required to incorporate security features into JTAG, owing to the small test interface available in most applications. Similarly, various protocols using ECC may been used. We chose the Schnorr protocol as it is provably secure and allows efficient implementation on space-constrained hardware.

An added positive side-effect of Schnorr is that it is "zero-knowledge" and thus no information about the secret key of the prover leaks during a protocol run. The zero-knowledge property may be useful in an uncontrolled in-the-field code update, debug or test environment where the communication channel between the test server and JTAG is untrusted and the secret need not be shared or linked to a communicating entity. Moreover, Schnorr is a very established protocol, and is used in Radio Frequency Identification (RFID) protocols [16, 17]. The related ECC-based Schnorr authentication protocol [13] is described in appendix A.

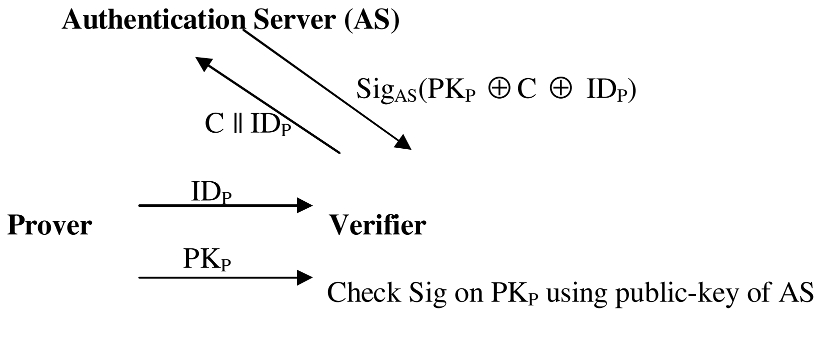
4.3 Public Key Verification

When using public key cryptography for authentication purposes, it is essential to verify the authenticity of the prover's public key. Traditionally, the link between a user's public key and some identifier of the user is captured in a digital certificate that is signed by a trusted third party (e.g., certificate authority or CA). By verifying this certificate, a verifier is assured that the public key that is provided by the prover is genuine. This means that it is sufficient to have a copy of the CA's public key in order to verify all public keys that are certified by the CA. It is clear that storing a single CA's public key is far more practical than storing a collection of symmetric keys that are shared with each possible prover. Therefore, we argue that our protocol, although more resource consuming, does provide a more practical solution when compared with previous JTAG authentication mechanisms that are based on symmetric cryptography only.

We propose two modes of operation, one is purely offline and the other uses an online connection to a trusted Authentication Server (AS).

In the offline mode, we assume that every prover has a certified public key and this certificate is signed by a trusted CA. Every verifier has a copy of the CA's public key. Before the actual Schnorr authentication protocol, the prover sends his certificate to the verifier. The verifier simply uses the CA's public key to verify the certificate. In case, the verifier has access to a "clock", the verifier can also check an optional expiration date inside the certificate. In case the JTAG device is the verifier, this clock will probably not be available and no expiration date can be verified. Note that in this scenario, it is not possible to revoke certificates, as it is not possible to use an online server to obtain revocation lists or use an Online Certificate Status Protocol (OCSP) like protocol.

In case the verifier has the possibility to contact the online trusted AS, we propose to use a simplified version of the OCSP protocol. The protocol steps are depicted in the next figure:


Authentication Server (AS)
$Sig_{AS}(PK_P \oplus C \oplus ID_P)$
$C \parallel ID_P$
ID_P
Prover
Verifier
PK_P
Check Sig on PK_P using public-key of AS

Sig_{AS} denotes signature of the prover's public key with the private key of the Authentication Server, ID_P and PK_P are the Identity and Public-key of the Prover respectively.

The prover starts with sending its ID_P and its public key PK_P to the verifier. The verifier then initiates a call to the AS by sending a fresh random challenge C and the ID of the prover to the AS. The AS will now lookup the public key of the prover, check whether it is still valid, and if so send a signature on the XOR of PK_P, C and ID_P back to the verifier. The verifier will only accept the public key received from the prover upon reception of a valid signature by the CA on the generated challenge $PK_P \oplus C \oplus ID_P$. We are XORing the ID, challenge and the public key (instead of appending) in order to make sure that we can sign this value without first using a cryptographic hash function. In case the messages we wish to sign becomes longer than the field length of the ECC module we use, we would first have to reduce this length by employing a cryptographic hash function (and potentially cropping the result). As the implementation of such a hash function would consume too much area, we have designed our protocol to operate without a hash function.

The signature scheme can be implemented using Elliptic Curve Digital Signature Algorithm (ECDSA). This consumes less area overhead than a 1024-bit RSA signature scheme. An area-efficient implementation is presented in [18]. In our implementation, we have modified the ECC Schnorr controller to allow ECDSA. The hashing involved in the ECDSA signature verification is avoided as we use 192-bit signatures (the same length as the message that is signed, which is the public-key of the prover). Through this public key certificate we protect the Schnorr protocol from man-in-the-middle attack too. Devices which have adequate resources (online connection to the authentication server) to support this authentication process can opt for an online mode, while other devices can have an offline mode of authentication.

In this paper, we provide two different implementations. One is over projective coordinates and another is over affine coordinates. In the first design, we do not implement an inversion module whereas it is included in the second design. Due to projective coordinates the first design invokes very few inversions which are performed iteratively on a multiplier unit following Fermat's little theorem. However, in affine coordinates inversion is performed at every iteration of point multiplication algorithm. Thus, a dedicated inversion unit based on extended Euclidian algorithm is implemented which also helps efficient execution of ECDSA on our secure JTAG scheme. Here we provide implementation details for both designs which provide better design variations and the user can opt for one of them in practice.

5. Secure JTAG Implementation

5.1 Integration of the ECC-processor with JTAG

An important contribution in our paper is the integration of the ECC based Schnorr controller and ECC point multiplier with the JTAG interface along with the other modules. This has been done in a seamless manner so as not to affect the timing aspects of the IEEE 1149.1 JTAG standard, and also keeping the behavior of the TAP finite state machine (illustrated in Appendix F) unchanged.

Our proposed architecture is shown in Fig. 1. The ordinary JTAG circuitry is enclosed within dotted lines, and it is divided into its two main components: the TAP finite state machine and the instruction decoder. The Schnorr protocol (described in Appendix A), as well as the ECDSA signature authentication are performed by the Schnorr controller, placed in the center of Fig. 1. It interacts with a modified JTAG instruction decoder, ECC module, and a 192-bit random number generator (a Linear Feedback Shift Register). The base point coordinates (curve parameters) are fetched from an external non-volatile memory.

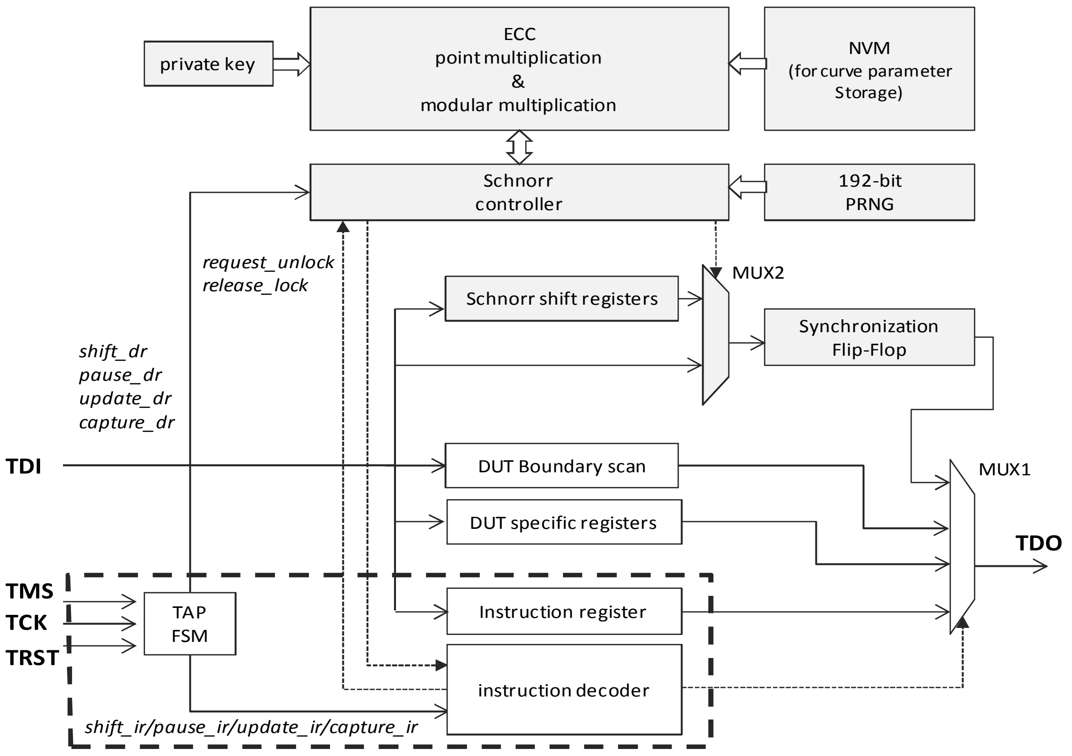


Fig. 1. JTAG-ECC controller Integration Architectural Block Diagram

The system is supposed to be locked in the beginning. In order to unlock it, the tester must manipulate the JTAG inputs to enter the new ‘UNLOCK’ instruction. Then, the instruction decoder informs the Schnorr controller to start the protocol, by means of the ‘request_unlock’ signal. As soon as the authenticity of the test server is verified, the Schnorr controller activates the ‘release_unlock’ signal, informing the instruction decoder that other instructions can now be performed. For instance, if the system is unlocked, the design under test (DUT) boundary scan register can be controlled. Meanwhile, when ‘release_unlock’ signal is not active, the instruction decoder sets the multiplexer ‘MUX1’ to always select the output from the multiplexer ‘MUX2’, which is controlled by the Schnorr controller, impeding the shift out of any DUT specific register.

During the protocol execution, the communication with test server consists of using the Schnorr shift registers (192 bits) to shift in and out information required for the protocol. For instance, the transmission of the intermediate values, ‘T_a’ and ‘T_b’ (Protocol in Appendix A) is performed by means of shifting out the value ‘T_a’ (or ‘T_b’) once the ECC point multiplication is finished. It is important to notice that the shifting is always controlled by the test server, and that the timing for executing point multiplications depends on the scalar multiplier. It means that the Schnorr controller must inform the test server that it has finished each operation of the protocol. This synchronization is achieved by always adding one flip-flop at the end of the Schnorr shift register that is set to ’1’ if the information in the shift register is valid, otherwise the multiplexer ‘MUX2’ selects the TDI input and the synchronization flip-flop is set to ‘0’. Thus, the test server keeps on shifting at least this one bit to detect that the Schnorr controller is ready for receiving the next data. The step-wise detailed ECC-based Schnorr protocol for Secure JTAG is described in Appendix B.

5.2 Implementation of the ECC processor

The exponentiations involved in the Schnorr protocol may be implemented using RSA or ECC. However, ECC involves much smaller bit lengths compared to RSA and is efficient in hardware. Hence we implement the Schnorr protocol using 192-bit ECC over prime fields which offers higher security compared with 1024-bit RSA. Highly efficient ECC and ECDSA implementations for contrained environments can be found in [28][29]. However, in this work, we present two new designs which are optimized both for area and timing suited for integration with the standard JTAG.

We use the 192-bit NIST ECC curve P192 and work in prime fields (F_p). The curve parameters used in our ECC implementation is as follows [19]:

p : The order of the prime field F_p.
a,b: The coefficients of the elliptic curve $y^2 = x^3+ax+b$.
n: The (prime) order of the base point P.
h: The cofactor.
x, y: The x and y coordinates of P.

P-192: $p = 2^{192}-2^{64}-1$, $a =-3$, $h = 1$
b = 0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1
n = 0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831
x = 0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012
y = 0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

The point operations over affine and projective coordinates are performed by standard formula taken from the literature, and are provided in Appendix C. In the projective coordinate, we represent a point as: (x=X/Z, y=Y/Z, c=1, d=1). In general, projective coordinates are introduced to avoid the relatively more costly inversion used in point-operation over affine coordinates. However, relativity among the costs of multiplication and inversion in F_p varies on their implementations. For example, when modular multiplication is computed in a bit serial fashion, it leads to $\log_2 p$ number of iterations (clock cycles); whereas, when inversion is performed by binary Euclidean algorithm, it requires at most $2 \log_2 p$ number of iterations (clock cycles). Following the above design technique, the point operations over affine coordinates outperform projective coordinates.

5.2.1 Design I: ECC over Projective Coordinates

In this implementation, we use the 1998 Cohen–Miyaji–Ono mixed coordinates for point addition [20] and the 2007 Bernstein–Lange formulae [21] for point doubling from Explicit Formula database for Short Weierstrass curves [22]. The equations are mentioned in Appendix C. To reduce area overhead, the adder and the Montgomery multiplier used in ECC have been optimized. The ordinary adder/subtractor (required for the intermediate operations of the Montgomery Multiplier) has been combined with the modular adder/subtractor (required for the addition/subtraction operations used to implement ECC in projective coordinates) using a 2-bit select signal. This helps reduce the area overhead further. The modules employed in our design are described below.

Schnorr & ECDSA controller modules: Figure 2 shows the block diagram of the Design I implementation.

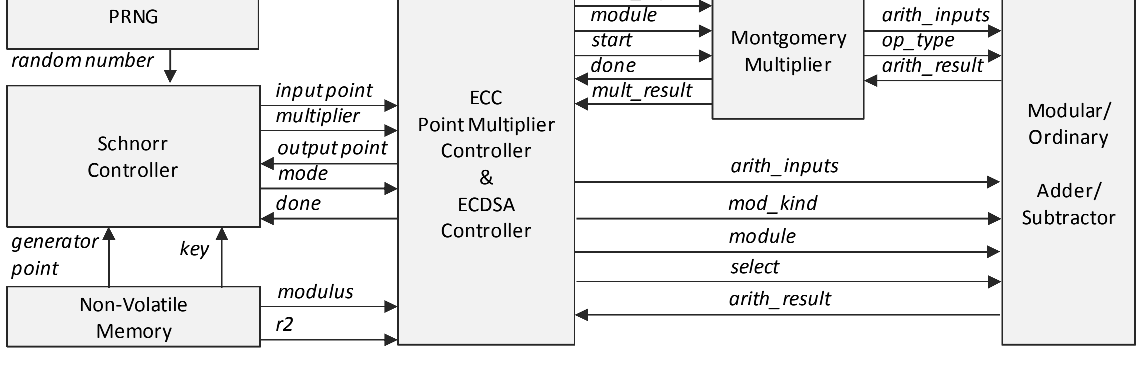


Fig. 2. Block diagram of the security architecture

The Schnorr protocol consists of two main operations: ECC point multiplication and modular multiplication. In order to perform an ECC point multiplication, the Schnorr controller loads correctly the inputs of the ECC & ECDSA controller, and sets the "mode" signal to '0'. Then it disables the "reset" signal of the ECC & ECDSA controller so it can initiate the execution. Then the Montgomery multiplier and the modular/ordinary adder/subtractor blocks are used to perform the point multiplication. As it can be seen, the adder/subtractor block is shared between the ECC controller and the Montgomery multiplier. If the ECC controller is using it, it sets the "select" signal to '1' and then it chooses the operation type by setting the "mod_kind" signal ('0' for addition and '1' for subtraction). Each ECC scalar multiplication is performed using the Montgomery Powering Ladder algorithm, which is also protected against Simple Power Analysis (SPA) attacks.

On the other hand, to perform a modular multiplication, we reuse the Montgomery multiplier. For that purpose we use the order of the prime number as modulus instead of the prime number itself. The Schnorr controller sets the mode to '1' and then uses the ECC controller as an interface to the Montgomery multiplier block. This interfacing was implemented in order to reuse the ECC controller finite-state-machine.

The ECDSA operation is performed partially by the ECC & ECDSA controller and partially by the Schnorr controller. It first executes all the ECDSA steps which require only integer multiplications by setting the mode to '2' and loading the signature into the ECDSA block. Then the Schnorr controller saves these intermediate values and reuses the ECC controller block to run the two final point multiplications. For executing the inversion present in the ECDSA protocol we used the Itoh-Tsujii algorithm [23] based on the Fermat's little theorem that allows to execute inversions using a modular multiplier. The Pseudo-Random Number Generator (PRNG) in the diagram which generates the random numbers required in the Schnorr protocol is a 192-bit LFSR. We are reseeding the LFSR after every authentication execution, with a new seed to avoid starting it with the same initial value on power up, in order to prevent replay attacks. Efficient LFSR reseeding techniques [30, 31, 32] using seed storage methods or seed derivation from the modules of the design can be used for the purpose. For security, the LFSR length must be large enough to prevent brute-force attacks (192-bit in our design) and irreducible polynomials used for the feedback taps to have all possible sequences ($2^{192} - 1$, in our case). Moreover, the reseeding must be done quite often to prevent prediction of generated sequences (at the beginning of every authentication as in our case). The new seed value is loaded into the LFSR as soon as the 'request_unlock' signal in Figure 1 goes high. Alternatively, for enhanced security, True Random Number Generators (TRNGs) based on Fibonacci or Galois Ring Oscillators [27], which have similar area overhead as LFSRs and substantially high randomness and unpredictability properties, can also be employed.

Montgomery multiplier. Montgomery's algorithm is the most common method for a fast implementation of modular multiplications.

Algorithm 1 in Appendix D presents an efficient implementation of this algorithm. As one can notice, the final comparison is optimized exploiting carry-save-adders (CSA). CSAs are used for the intermediate computations and then a full addition is performed to convert the final carry-save result into a conventional form, such as presented in the Algorithm 2, Appendix D, and Figure 3.b. The CSA adders have indeed a small area and avoid carry propagation, i.e. are computed in constant time independently of the operands' length.

However, as a modular adder/subtractor is needed for ECC, we have decided to use the initial algorithm. We have indeed modified the adder/subtractor block to have an ordinary addition/subtraction also, in order to use this block in our Montgomery multiplication implementation. Optimizing the area is indeed our main objective, so using existing resources is better than implementing CSA adders. As a consequence, our implementation takes more or less twice as many cycles, but it is the one that optimizes most of the area. In the end, we have managed to optimize the area by more than 16%, in comparison with the RTL description of the original design with an unoptimized implementation of the adder/subtractor. This optimized arithmetic block is presented in Apendix E.

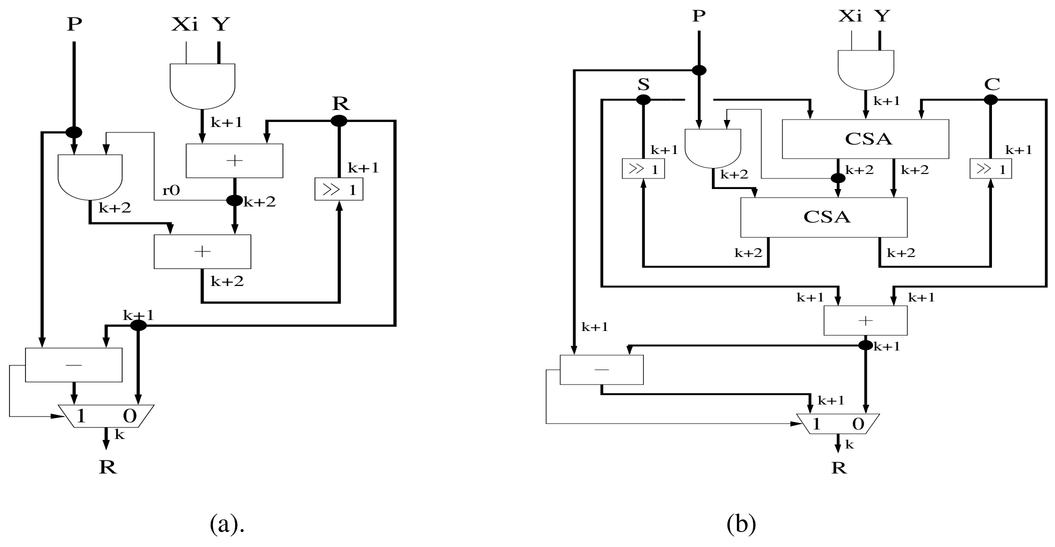


Fig. 3. Montgomery multiplier. a: Classical architecture b: CSA architecture

5.2.2 Design II: ECC over Affine Coordinates

The execution of the Schnorr protocol and ECDSA consists of several finite field operations (including inversion) and operations on elliptic curves. In Weierstrass elliptic curve, a point is primarily defined over Affine (x, y) coordinates which is further redefined over several Projective coordinates with the help of a third variable (X, Y, Z) in order to avoid inversion in point operations performed by chord-and-tangent method. Explicit formulae are provided in Appendix C. A single inversion is eliminated by several (4-12) multiplications in Projective coordinates – still research is going on for finding coordinate systems to lower down multiplications in a point operation.

However, the implementation technique also plays an important role for improving efficiency of elliptic curve operations under a resource constrained environment like JTAG. It is already described in Section 5.2 that delay of a binary inversion/division method is just twice that of a bit-serial multiplication where both of them are assumed to be implemented by simple adder circuits – demands area in the same decimal order (3 times). On the other hand, efficient implementation of modular (Montgomery) multiplication could be achieved through digit-serial (parallel) architecture which demands much more area (order of digit length) and may not be affordable in the application of secure JTAG implementation. Thus design II attempts to implement a compact and flexible architecture for executing Schnorr protocol and ECDSA over Affine coordinates. Besides, this design computes all modular operations directly on 2's complement binary domain that avoids cost of domain conversions compared to the first design.

Flexible Datapath. In order to reduce the complexity of the controller logic, design II consists of a flexible datapath having all arithmetic blocks. There are two top level controllers in the current secure JTAG implementation – namely ECDSA-controller and Schnorr-controller. These controllers generate instructions like *PointAdd, PointMult, FieldAdd, FieldMult, FieldInv, FieldSub.* In the next lower level, there is an ECMULT-controller which primarily generates two instructions - *PointAdd* and *PointDbl.* All instructions generated by top level controllers are first checked by the ECMULT-controller which further passes through the next lower level. Except *PointMult*, all other instructions are directly executed by the datapath shown in Fig. 4. The instruction *PointMult* consists of *PointAdd* and *PointDbl* instructions which are generated in proper sequence by the ECMULT-controller. All controller logic in design II are realized as finite state-machines in which the final state sends a *done* signal to its predecessor. With the help of five temporary registers the datapath shown in Fig. 4 computes a point operation (point doubling or point addition) as a single instruction – in which case the output of an execution is stored and supplied back to the memory through x_3 and y_3 ports. On the other hand, the outputs for all other finite field operations are directly generated from individual arithmetic units. In order to execute *PointAdd* instruction the datapath takes the input data from 'x_1', 'y_1', 'x_2', 'y_2', 'p', and 'a' ports, whereas for executing individual finite field operation the ports are configured by the upper level controller logic.

Prime field Multiplication. In this design we use Blakley multiplication which is based on the iterative execution of doubling and addition. All internal operations are performed in respective prime field, that is intermediate results are always in their reduced form. Hence, the costly final reductions are eliminated. The multiplier unit contained in the datapath block (Fig. 4) computes a multiplication *ab mod p* in $\log_2$ p number of clock cycles, assuming that both a and b also have lengths of $\log_2$ p.

Prime field Inversion and Division. The prime field inversion and division could be efficiently computed by binary inversion division algorithm, which is based on binary Euclidian algorithm. The current design follows the implementation of such a unit that is described in [24]. The current module can compute one inversion (used on ECDSA) as well as one division (used to execute *PointAdd* and *PointDbl*) in 2 $\log_2$ p number of clock cycles.

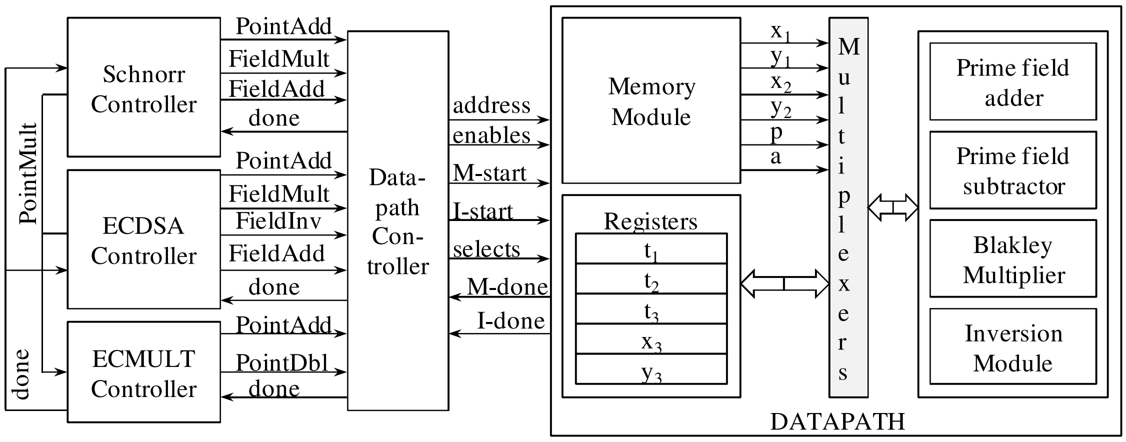


Fig. 4. Datapath of the design II

Design II executes a *PointAdd* instruction in 5 $\log_2$ p + 6 clock cycles and *PointDbl* in 4 $\log_2$ p + 8 clock cycles. The clock cycles required to execute a *PointMult* instruction is: $\log_2$ k * (4 $\log_2$ p + 8) + (#k-1) * (5 $\log_2$ p + 6), where k represents the scalar multiplier in kP and #k indicates the Hamming weight of k.

6. Results

We present here the area and timing results of our implementation. Both ASIC and FPGA results of the overall secure JTAG design with the sub-modules are mentioned. Though our design is larger than earlier methods, owing to the use of public-key cryptography (as opposed to symmetric-key usage in the other approaches), this helps solve the key-management problem inherent in other approaches to a great extent.

6.1 Area Overhead

The ASIC area requirements in terms of gate equivalents (GEs) (synthesized with Synopsys Design Compiler v2009.06 for a Faraday 130 nm library) for the modules used in our ECC implementation are given in the table below. The FPGA Synthesis results on Xilinx ISE 12.4 (with Virtex 6 xc6vlx75t family) for the modules are also presented.

Table 1: Hardware cost of secure JTAG

Module	**Design I ASIC (GEs)**	**Design II ASIC (GEs)**	**Design I FPGA (Slices)**	**Design II FPGA (Slices)**
Arithmetic unit (modular adder and subtractor)	1374	5128	164	311
Modular multiplier *	5152	7314	615	756
Inversion module	--	24313	--	1482
Controller and data multiplexers	40190	10295	2189	531
Total	46716	47050	2968	3080

*Montgomery multiplier for Design I, while Blakley multiplier for Design II

Hence, as shown in the table, we require a total of 46716 GEs for design I and 47050 GEs for Design II to implement the secure JTAG Scheme with the Schnorr and ECDSA controllers. We choose the solution described in [12] for having an estimative of area overhead of our approach. The cost of that solution is 25k gates. It means that our solution is around twice larger than the solution in [12]. Our solution is based on public-key cryptography, which inherently demands more hardware resources than symmetric-key based approaches. The area calculations do not consider the overhead of the Hash function implementation which takes around 10k gates [33] for SHA-1, in case the message lengths become longer than the ECC field size (as mentioned in Section 4.3). However, this is not applicable for properly designed protocols, as in our case. It must be also noted that all secure authentication schemes including the one proposed in this paper require Non-volatile memory for storing cryptographic keys.

The area requirement in our designs can be reduced further by making use of a tiny custom microcontroller with an Instruction Set Extension (ISE), as in [28]. Here only the top-level ECDSA commands are managed with a processor. Moreover, replacing the Montgomery Multiplier (suitable for general prime-field operations) with more efficient multipliers employing Mersenne-like NIST prime reduction suitable for prime fields over F_p (with the prime number p being a Pseudo Mersenne number as used in NIST curves) can also help reduce the execution time for an Elliptic Curve scalar multiplication.

There are of course much more compact implementations available in the literature, for instance, the 192-bit ECDSA implementation in [28] employing the same NIST recommended curve as in our case requires only 19.1 KGEs (thus consuming 23.5% less than the approach in [12]) and 859,188 cycles in total for the combined operations of ECDSA, Hash and Random number generation required for the protocol execution. Similarly the most area efficient 163-bit ECC implementation in [29] consumes only 12.5 KGEs (thus taking half the area in [12]) and 275,816 cycles for one Elliptic Curve scalar multiplication. In this paper, though we did not achieve such high compactness, we have shown the feasibility of integration of the JTAG with the ECC and ECDSA modules by presenting combined area and timing results which have limited overheads.

6.2 Timing overhead

The impact of the proposed solution in the use of the JTAG standard consists of an initial delay for executing the Schnorr protocol/ECDSA. Once the authentication and the signature verification steps are finished, the JTAG is unlocked and the JTAG instructions can be used without any timing overhead.

The initial delay is due to three main operations: 1) the time to request the unlock (associated with the time to insert the instruction 'UNLOCK') and the time to release the lock; 2) the time to shift in the protocol inputs and shift out the protocol outputs using the JTAG controller; and 3) the time to perform the protocol operations, including ECC point multiplications, ordinary multiplications and additional operations to communicate between the dedicated Schnorr protocol modules. The first two operation types are measured in test clock cycles that depend on the JTAG frequency, while the last operation type is measured in functional clock cycles, the functional clock being usually faster than the test clock. The timing overhead is presented in Table 2, where we have distinct four scenarios. The first scenario is a one-way authentication (manufacturer environment in Appendix A) in which the DUT acts as prover (A) and the test server acts as verifier (B). The only scalar (point) multiplication performed for A is $n_a.P$ for generating T_a. The second scenario is a one-way authentication (manufacturer environment in Appendix A), but the roles of prover and verifier are reversed. Here the DUT acting as the verifier B performs two scalar multiplications ($s.P$ and $n_b.P_a$). The third case is the two-way authentication (in the field update in Appendix A). Here, both A and B perform three scalar multiplications ($n_a.P$, $s_1.P$ and $n_a'.P_b$ for A, and $n_b'.P$, $s.P$ and $n_b.P_a$ for B). Finally, the last one is the timing overhead associated with the execution of the ECDSA signature verification, which requires two scalar multiplications.

Table 2. Detailed timing estimates

Scenario	Operation	# Clock cycles		Clock class
		Design I	Design II	
1. One way Authentication (DUT is the prover)	Unlocking	13	13	Test clock
	Time to shift data in and out	768	768	Test clock
	Protocol (1 k.P* operation)	3068150	240762	Functional clock
2. One way Authentication (DUT is the verifier)	Unlocking	13	13	Test clock
	Time to shift data in and out	768	768	Test clock
	Protocol (2 k.P* operations)	6136692	482130	Functional clock
3. Mutual(Two-way) Authentication	Unlocking	13	13	Test clock
	Time to shift data in and out	960	960	Test clock
	Protocol (3 k.P* operations each for prover and verifier)	9204842	722892	Functional clock
4. ECDSA	Unlocking	13	13	Test clock
	Time to shift data in and out	576	576	Test clock
	Protocol (2 k.P* operations)	6137075	482324	Functional clock

* 'k.P' indicates one Elliptic Curve Scalar Multiplication.

For having an estimation of time in milliseconds, we suppose a 100MHz clock frequency for the JTAG Test clock, and 115MHz as functional clock frequency for Design I and 123MHz for Design II, as shown in Table 3. The functional clock frequency is the maximum operating frequency obtained from FPGA synthesis. The test clock and the functional clock can be also the same without involving any design change. Considering the mutual authentication scenario with ECDSA signature verification, Design I has an initial delay of 133.42ms while Design II has an initial delay of 9.83ms.

Table 3. Time delay for authentication

	Functional Clock (MHz)	Delay for authentication (ms)#			
		Scenario 1	Scenario 2	Scenario 3	Scenario 4
Design I	115	26.67	53.37	80.05	53.37
Design II	123	1.96	3.93	5.89	3.94

# Delays are based on 100 MHz clock frequency for JTAG Test clock and respective Functional clock.

7. Conclusion

In this paper, we have presented the implementation of a secure test scheme integrating the provably secure Schnorr protocol with JTAG-based testing. The key management problem inherent in previous symmetric-key based approaches is overcome through the use of public-key cryptography in our test scheme. Moreover, we present detailed hardware implementations, area and timing results for our ECC and ECDSA-based authentication protocol. To the best of our knowledge, this is the first complete work for securing the JTAG interface using public-key cryptography which also provides mutual authentication between the device and the tester.

Acknowledgment

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish iMinds projects, and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. In addition, this work is supported in part by the Flemish Government, FWO G.0550.12N, by the Hercules Foundation AKUL/11/19, and by the European Commission through the ICT programme under FP7-ICT-2011-8 HINT. Amitabh Das was initially funded by the Erasmus Mundus External Cooperation Window Lot 15 (EMECW15) when part of the work was performed. Santosh Ghosh is a beneficiary of a mobility grant from the Belgian Federal Science Policy Office co-funded by the Marie Curie Actions from the European Commission.

References

1. IEEE Standard. 1149.1-1990 - IEEE Standard Test Access Port and Boundary-Scan Architecture, 1990.

2. IEEE P1687 and In-Circuit Test (ICT). Asset Intertech article, June 2011.

3. Maestra Comprehensive Test for Satellite Testing V5. www.maestra.ca.

4. Greenemeier, L.: iPhone Hacks Annoy AT&T but Are Unlikely to Bruise Apple. Scientific American, August 30, 2007.

5. Becher, A., Benenson, Z., and Dornseif, M.: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks. SPC 2006, LNCS 3934, pp. 104–118, 2006.

6. Hartung, C., Balasalle, J., and Han, R.: Node Compromise in Sensor Networks: The Need for Secure Systems. Technical Report CU-CS-990-05, Dept of Computer Science, Univ of Colorado at Boulder, 2005.

7. Spartan-3 Generation Configuration User Guide for Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families. UG332 (v1.6) October 26, 2009, pp. 80.

8. Novak, F., and Biasizzo, A.: Security Extension for IEEE Std. 1149.1. Journal of Electronic Testing: Theory and Applications 22, pp. 301–303, 2006.

9. Rosenfeld, K., and Karri, R.: Attacks and Defences for JTAG. IEEE Design and Test of Computers, 2010.

10. Clark, C.J.: Anti-tamper JTAG TAP design enables DRM to JTAG registers and P1687 on-chip instruments. IEEE Symposium on Hardware-Oriented Security and Trust (HOST) 2010.

11. Pierce, L., and Tragoudas, S.: Multi-level secure JTAG architecture. IOLTS(2011), pp. 208-209.

12. Park, K., Yoo, S.G., Kim, T., and Kim, J.: JTAG Security System Based on Credentials. Journal of Electronic Testing: Theory and Applications, September 2010.

13. Schnorr, C.P.: Efficient identification and signatures for smart cards. In G Brassard, ed. Advances in Cryptology – Crypto '89, pp. 239–252, LNCS 435, 1990.

14. Guide to Understanding JTAG Fuses and Security: An Intermediate Look at the AVR JTAG Interface. AVRFreaks.net, Sept 2002.

15. Rippel, E.: Security Challenges in Embedded Designs. Discretix Technologies Ltd., Design & Reuse article. http://www.design-reuse.com/articles/20671/security-embedded-design.html.

16. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: An Elliptic Curve Processor Suitable For RFID-Tags. IACR Cryptology ePrint Archive, 2006.

17. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., and Verbauwhede, I.: Public-Key Cryptography for RFID-Tags. Workshop on RFID Security, pp. 61-76, 2006.

18. Kern, T., and Feldhofer, M.: Low-Resource ECDSA Implementation for Passive RFID Tags, ICECS 2010.

19. Hankerson, D., Menezes, A., and Vanstone, S.: Guide to Elliptic Curve Cryptography, pp. 262, Sample parameters.

20. Cohen, H., Miyaji, A., and Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. ASIACRYPT '98. LNCS 1514, pp. 51-65, 1998.

21. Bernstein, D.J., and Lange. T.: Faster addition and doubling on elliptic curves. ASIACRYPT 2007. LNCS 4833, pp. 29-50, Springer, 2007.

22. Explicit Formula Database. http://www.hyperelliptic.org/EFD/g1p/auto-shortw.html.

23. Itoh, T., and Tsujii, S.: A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. Information and Computation, 78: pp. 171-177, 1988.

24. Ghosh, S., Mukhopadhyay, D., and Roychowdhury, D.: Petrel: power and timing attack resistant elliptic curve scalar multiplier based on programmable arithmetic unit. IEEE Transactions on Circuits and Systems I, Vol 58, No. 11, pp. 1798–1812, 2011.

25. Alam, M., Ray, R., Mukhopadhayay, D., Ghosh, S., Roychowdhury, D., Sengupta, I.: An Area Optimized Reconfigurable Encryptor for AES-Rijndael, DATE 2007, pp. 1116 - 1121.

26. Buskey, R.F., and Frosik, B.B.: Protected JTAG, Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06), 0-7695-2637-3/06.

27. Jovan Dj. Golic, "New Methods for Digital Generation and Postprocessing of Random Data", IEEE Transactions on Computers, Vol. 55, No. 10, October 2006.

28. Michael Hutter, Martin Feldhofer, Thomas Plos, "An ECDSA Processor for RFID Authentication", RFIDSec LNCS 2010, Volume 6370, 2010, pp 189-202.

29. Yong Ki Lee, Kazuo Sakiyama, Lejla Batina, Ingrid Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID", IEEE Transactions on Computers, November 2008 (vol. 57 no. 11), pp. 1514-1527.

30. Stelios Neophytou, Maria K. Michael, Spyros Tragoudas, "Efficient Deterministic Test Generation for BIST Schemes with LFSR Reseeding", 12th IEEE International On-Line Testing Symposium, 2006 (IOLTS'06).

31. Zhanglei Wang, Krishnendu Chakrabarty, and Seongmoon Wang, "Integrated LFSR Reseeding, Test Access Optimization, and Test Scheduling for Core-Based System-on-Chip", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 28, No. 8, August 2009.

32. Mahmut Yilmaz and Krishnendu Chakrabarty, "Seed Selection in LFSR-Reseeding-Based Test Compression for the Detection of Small-Delay Defects", DATE 2009.

33. A. Satoh and T. Inoue, "ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS," Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05), 2005

Appendix A

ECC-based Schnorr Protocol

In the manufacturer environment scenario, A is the prover (Secure JTAG) and B is the verifier (Test server):

P_a is the public key of A and k_a is the private key of A, which are related by:
$P_a = k_a.P$
where P is the initial point on the Elliptic curve (base point), which is public. $k_a.P$ represents a point multiplication of scalar k_a with base-point P.

Goal: B wants to be ensured the identity of A, in other words A knows k_a.

Protocol:
1) A generates a random number n_a and sends an intermediate value 'T_a' (point multiplication of n_a and P) to B;

$A \rightarrow B$: $T_a = n_a.P$

2) B generates a random number n_b and sends it to A;

$A \leftarrow B$: n_b

3) A sends 's' to B;

$A \rightarrow B$: $s = n_a + k_a.n_b$

Here $k_a.n_b$ represents an integer multiplication, while '+' indicates an ordinary addition.

B can verify that A is A by calculating the point multiplication of scalar s with base-point P and cross-checking it with the modular addition of 'T_a' with the point multiplication of n_b and P_a:

$$s.P = T_a + n_b.P_a$$
$$(n_a + k_a.n_b)P = (n_a.P) + (k_a.P).n_b$$
$$n_a.P + k_a.n_b.P = n_a.P + k_a.n_b.P$$

Thus B verifies the identity of A by only knowing A's public key P_a.

For in-the-field updates, debug and test:

A is the prover (Secure JTAG), when B is the verifier (Test server).
B is the prover (Test server), when A is the verifier (Secure JTAG).

P_a is the public key of A and k_a is the private key of A, which are related by:
$P_a = k_a.P$, where P is the initial point on the Elliptic curve (base point), which is public.

P_b is the public key of B and k_b is the private key of B, which are related by:
$P_b = k_b.P$, where P is the initial point on the Elliptic curve (base point), which is public.

Goal: B wants to be sure that A is actually A, in other words, that A knows k_a. Similarly, A wants to be sure that B is actually B, in other words, that B knows k_b.

Protocol:
1) A generates a random number n_a, and sends it along with an intermediate value 'T_a' to B, which is calculated as:

$$A \rightarrow B: \quad T_a = n_a.P$$

2) B generates two random number n_b and n_b', and sends n_b along with an intermediate value 'T_b' to A, which is calculated as:

$$A \leftarrow B: \quad T_b = n_b'.P,\ n_b$$

3) A generates another random number n_a' and sends it along with sends 's' to B, B sends 's_1' to A:

$$A \rightarrow B: \quad s = n_a + k_a.n_b,\ n_a'$$
$$A \leftarrow B: \quad s_1 = n_b' + k_b.n_a'$$

B can verify that A is A by calculating:

$$s.P = T_a + n_b.P_a$$
$$(n_a + k_a.n_b).P = (n_a.P) + n_b.(k_a.P)$$
$$n_a.P + k_a.n_b.P = n_a.P + n_b.k_a.P$$

Similarly, A can verify that B is B by calculating:

$$s_1.P = T_b + n_a'.P_b$$
$$(n_b' + k_b.n_a').P = (n_b'.P) + n_a'.(k_b.P)$$
$$n_b'.P + k_b.n_a'.P = n_b'.P + n_a'.k_b.P$$

Thus B verifies the identity of A by only knowing A's public key P_a, and A verifies the identity of B by only knowing B's public key P_b.

Moreover, $n_a.n_b'.P$ can be used as a session key K to encrypt all future communication between the security chip and test server. The reason behind this is that A knows $n_a.P$ and n_b', while B knows n_a and $n_b'.P$ from which they can construct K, but any unauthorized party cannot do so. This may be particularly useful for instance, in the case of pay-TV updates happening on the set-top box from a remote server using a network communication, where an eavesdropper can listen to the channel in between.

Appendix B

ECC based Schnorr for secure JTAG

The execution of the Schnorr protocol is now explained in some detail using the block diagram below:

1) First, the JTAG public key Pa is calculated. For this, the ECC controller module sends the private JTAG key k_a (from on-chip storage) and the base point coordinates and other curve parameters (prime number, R*R mod n) from the non-volatile memory to the ECC point multiplier module. It then instructs the point multiplier module to start an ECC point multiplication operation.
2) The ECC point multiplier then performs a point multiplication of the scalar k_a with the base point P and returns the result (P_a) back to the ECC controller module. This result is stored in a 192-bit temporary register inside the controller module.
3) A 192-bit random number n_a is generated by the on-chip random-number generator and sent to the ECC controller module.
4) The ECC controller module then sends this n_a and the base point coordinates and other curve parameters from the non-volatile memory to the ECC point multiplier module. It then instructs the point multiplier module to start an ECC point multiplication operation.
5) The ECC point multiplier then performs a point multiplication of the scalar n_a with the base point P and returns the result (‘T_a’) back to the ECC controller module. This result is stored in another temporary register inside the controller module.
6) The test server then generates a 192-bit random number n_b and sends this to the JTAG module bit-by-bit through the TDI input. This is then stored in the 192-bit shift (data) register of the JTAG.
7) n_b and the private key of the JTAG (k_a) is transferred to the ECC.
8) For the integer multiplication of k_a with n_b, the ECC controller instructs the arithmetic module inside the point multiplier module to perform a modular multiplication of k_a with n_b using the ‘order of the prime’ (fetched from the non-volatile memory storage of curve parameters) as the modulus (this is equivalent to integer multiplication of k_a with n_b). The result is stored back in a 192-bit register inside the ECC controller module.
9) A modular addition of n_a with $k_a.n_b$ is then performed in the arithmetic block inside the point multiplier module. For this, the appropriate control is provided from the ECC controller which also stores the result of the computation (‘s’) in the same 192-bit register.
10) The ECC controller module then sends ‘s’ and the base point coordinates and other curve parameters from the non-volatile memory to the ECC point multiplier module. It then instructs the point multiplier module to start an ECC point multiplication operation.
11) The ECC point multiplier then performs a point multiplication of the scalar ‘s’ with the base point P and returns the result back to the ECC controller module. This result is stored in the same register inside the controller module.
12) Next, the ECC controller module then sends n_b and the public key of the JTAG (P_a) and other curve parameters from the non-volatile memory to the ECC point multiplier module. It then instructs the point multiplier module to start an ECC point multiplication operation.
13) The ECC point multiplier then performs a point multiplication of the scalar n_b with P_a and returns the result back to the ECC controller module. This result is stored in another temporary register inside the controller module.
14) A modular addition of the stored ‘T_a’ with $n_b.P_a$ is then performed in the arithmetic block inside the point multiplier module. For this, the appropriate control is provided from the ECC controller which also stores the result of the computation in the same 192-bit register.
15) The result of the above computation ($T_a + n_b.P_a$) is then compared with s.P computed and stored earlier inside the comparator module in the ECC controller module. If they match, then only the JTAG is allowed to enter the test and debug modes, otherwise it remains in the bypass mode.

Appendix C

Point Addition and Point Doubling in Affine Coordinates:

When P = (x_P,y_P) and Q = (x_Q,y_Q) are not negative of each other, then P + Q = R where

$$s = (y_P - y_Q) / (x_P - x_Q)$$
$$x_R = s^2 - x_P - x_Q \text{ and } y_R = -y_P + s(x_P - x_R)$$

Note that s is the slope of the line through P and Q.

Similarly, When y_P is not 0, then 2P = R where

$$s = (3x_P^2 + a) / (2y_P)$$
$$x_R = s^2 - 2x_P \text{ and } y_R = -y_P + s(x_P - x_R)$$

Recall that a is one of the parameters chosen with the elliptic curve and that s is the tangent on the point P.

Formulae for ECC Point Addition and Doubling in Projective Coordinates:

Table C1: Explicit Formulae:

Point Addition	Point doubling
Cost	
12 Field Multiplications + 2 Squarings + 6 additions + 1 shift.	7 Multiplications + 3 Squarings + 5 additions + 4 shifts + 1 cubing.
Source	
1998 Cohen–Miyaji–Ono [24] "Efficient elliptic curve exponentiation using mixed coordinates"	2007 Bernstein–Lange [23]
Formulae	

Y1Z2 = Y1*Z2
X1Z2 = X1*Z2
Z1Z2 = Z1*Z2
u = Y2*Z1 - Y1Z2
uu = u*u
v = X2*Z1 - X1Z2
vv = v*v
vvv = v*vv
R = vv*X1Z2
A = uu*Z1Z2 – vvv - 2*R
X3 = v*A
Y3 = u*(R - A) - vvv*Y1Z2
Z3 = vvv*Z1Z2
	w = 3*(X1 - Z1)*(X1 + Z1)
	s = 2*Y1*Z1
	ss = s*s
	sss = s*ss
	R = Y1*s
	RR = R*R
	B = 2*X1*R
	h = w*w - 2*B
	X3 = h*s
	Y3 = w*(B - h) - 2*RR
	Z3 = sss

Here '*' indicates modular multiplication which in our case has been implemented using the Montgomery Multiplier. The addition and subtraction operations denoted here are all modular in nature. Using these set of formulae have the additional advantage that the computations are not dependent on the value of parameters 'a' and 'b'.

Appendix D

Algorithm 1:
Modified Montgomery modular multiplication

```
Input: A, B, M
Output: $R = X\,Y\,2^{-(n+2)} \bmod M$
  $a_i$ : i^th bit of A, $s_0$ : LSB of S
    1. S = 0, C = 0;
    2. for i=0 to n+1
         S, C = S + C + $a_i$ x B;
         S, C = S + C + $s_0$ x M;
         S = S div 2;
         C = C div 2;
    3. R = S + C
    4. if $R \geq M$ then R=R-M
    5. return R
```

Algorithm 2:
Montgomery modular multiplication

```
Input: A, B, M
Output: $R = X\,Y\,2^{-(n+2)} \bmod M$
  $a_i$ : i^th bit of A
    $r_0$ : LSB of R
    1. R = 0;
    2. for i=0 to n-1
         R = R + $a_i$ x B;
         R = R + $r_0$ x M;
         R = R div 2;
    3. if $R \geq M$ then R=R-M
    4. return R
```

Appendix E

Modular adder / subtractor. A "naïve" implementation of a modular addition A+B mod P is presented in Fig. A1.a; it consists in computing A+B, and then subtracting P to this result. A comparison between these two intermediate results allows choosing which one to use for the final result. However, this comparator could be avoided by observing the carry (borrow) out signal of addition (subtraction) which could be realized by a single OR gate (instead of a 192-bit comparator) such as presented in Fig. A1.b. Concerning the subtraction, the principle is the same: computing A-B and then A-B+P, and comparing these intermediate results to choose which one to use for the final result. A naïve and an optimized version of the subtraction are presented in Fig. A1.c and A1.d.

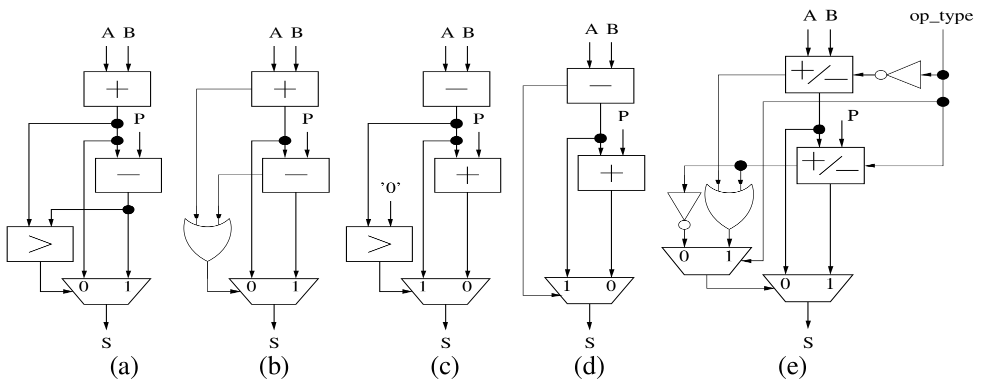


Fig. A1. Modular addition and subtraction implementations

The two optimized versions (Fig. A1.b and A1.d) have been combined to produce an optimized modular adder/subtractor block such as depicted in Fig. A1.e. In this architecture an input (op_type) is used to generate whether an addition or a subtraction (put to 1 for an addition and 0 for a subtraction). This architecture uses two adder/subtractor blocks (i.e. an addition combined with the inversion (or not) of the second operand using XOR gates and the input carry to '1' (or '0')) and the optimized comparison implementation depicted earlier. Concerning the architecture used for the additions/subtractions, we have used the library provided by the synthesizer which includes highly optimized RTL for arithmetic building blocks.

In the end, an efficient adder architecture combined with an optimized comparison implantation have led us optimize the area of more than 90%, by comparison with the area obtained from a VHDL file directly generated by our Gezel implementation.

Appendix F

16-cycle JTAG TAP Controller State Diagram

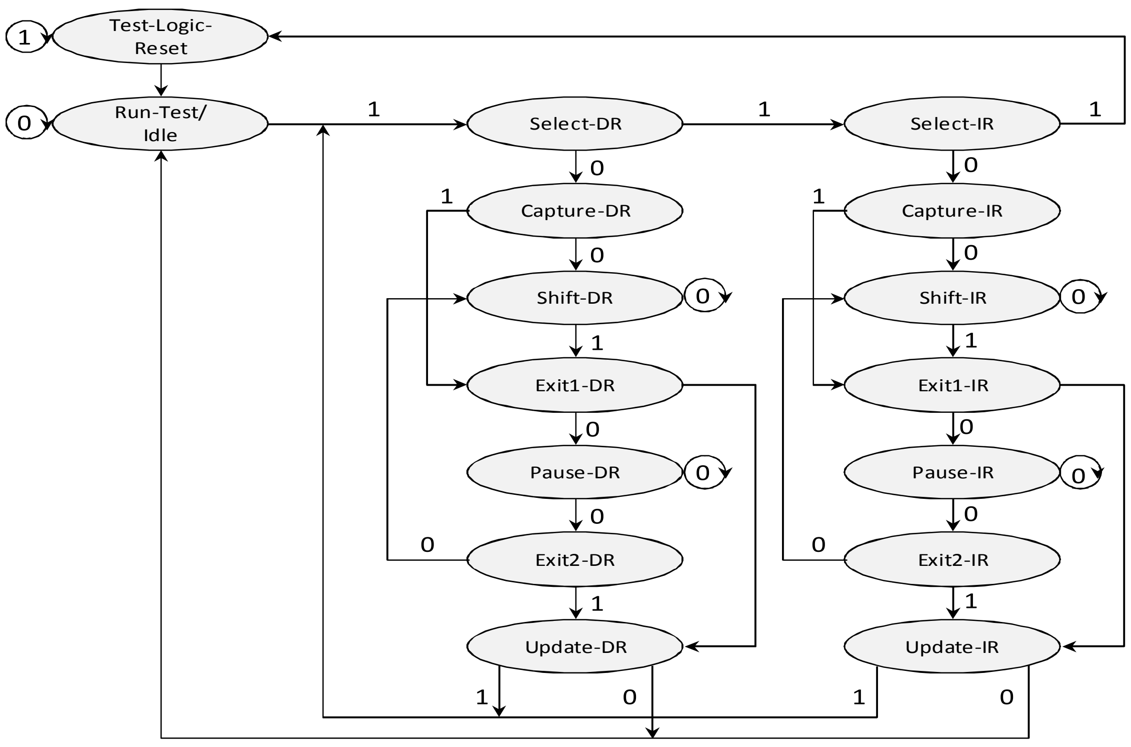


Fig. A2. TAP controller state diagram