Anonymous Attestation for IoT

Santosh Ghosh, Andrew H. Reinders, Rafael Misoczki, and Manoj R. Sastry

Intel Labs, Intel Corporation, USA
{santosh.ghosh, andrew.h.reinders, rafael.misoczki,
manoj.r.sastry}@intel.com

Abstract. Internet of Things (IoT) have seen tremendous growth and are being deployed pervasively in areas such as home, surveillance, healthcare and transportation. These devices collect and process sensitive data with respect to user's privacy. Protecting the privacy of the user is an essential aspect of security, and anonymous attestation of IoT devices are critical to enable privacy-preserving mechanisms. Enhanced Privacy ID (EPID) is an industry-standard cryptographic scheme that offers anonymous attestation. It is based on group signature scheme constructed from bilinear pairings, and provides anonymity and sophisticated revocation capabilities (private-key based revocation and signature-based revocation). Despite the interesting privacy-preserving features, EPID operations are very computational and memory intensive. In this paper, we present a small footprint anonymous attestation solution based on EPID that can meet the stringent resource requirements of IoT devices. A specific modular-reduction technique targeting the EPID prime number has been developed resulting in 50% latency reduction compared to conventional reduction techniques. Furthermore, we developed a multi-exponentiation technique that significantly reduces the runtime memory requirements. Our proposed design can be implemented as SW-only, or it can utilize an integrated Elliptic Curve and Galois Field HW accelerator. The EPID SW stack has a small object code footprint of 22kB. We developed a prototype on a 32-bit microcontroller that computes EPID signature generation in 17.9s at 32MHz.

Keywords: Enhanced Privacy ID, EPID, Authentication, Attestation, Bilinear Pairing, Optimal-ate Pairing

1 Introduction

Enhanced Privacy ID (EPID) was introduced by Brickell and Li [11] in 2007 to address the need for anonymous attestation. The authentication technique in EPID is based on Boneh, Boyen, and Shacham's group signature scheme [6] and Furukawa and Imai's group signature scheme [20]. One of the main features of EPID consists of its sophisticated revocation techniques which are constructed from pairing-based cryptography [10, 13, 14].

The EPID scheme follows the privacy-preserving technique in digital signatures that was introduced by Camenisch and Lysyanskaya in 2001 [16] as the generalization of the anonymous credential system due to [15] and the generalization of the group signature scheme proposed by Ateniese et. al [2]. With Strong-RSA assumption, [16] provides the provably secure signature scheme where a member can participate in a group and can generate a signature without revealing its identity. Later in 2004, this scheme was extended to Direct Anonymous Attestation (DAA) [9], which was adopted by the Trusted Computing Group (TCG) as the method for remote attestation in Trusted Platform Module (TPM). The DAA scheme has evolved in the past several years to achieve higher security, improved efficiency and fine-grained revocation capabilities [10–14]. This resulted in EPID being adopted as an ISO standard in 2013 [18].

The EPID scheme as described in [12] is based on bilinear pairings on Barreto-Naehrig (BN) elliptic curves [4], which are pairing-friendly elliptic curves. It uses Optimal-ate pairing [31] on BN curves comprised of three bilinear groups: G_1, G_2, and G_T. In EPID scheme, each of these groups are defined with a 256-bit prime ordered sub-groups where, G_1 is defined over a 256-bit BN curve on the base field (F_q), G_2 is defined on BN curve with sextic twist on F_{q^2} so that a member of G_2 is defined as the tuple (x, y) where $x, y \in F_{q^2}$ and the group G_T is defined on $F_{q^{12}}$ as an integer group. The EPID Sign and Verify operations involve multiple Optimal-ate pairing operations as well as multiple exponentiations on G_1 and G_T which are essentially computed on 256-bit and 3072-bit numbers. Therefore, a naïve implementation of the EPID protocol incurs significantly high latency, compute energy and area/memory footprint overhead which are the bottlenecks for enabling EPID on resource constrained IoT devices.

1.1 Contributions

To address the above problem, we introduce an optimized SW implementation of EPID scheme for resource-constrained embedded IoT platforms. A set of novel techniques has been explored, implemented and carefully integrated resulting in the proposed EPID software stack with minimal code and memory footprint, as well as it keeps up an acceptable latency on 32-bit microcontrollers. Additionally, there is a compile-time knob that replaces the specific SW modules by a driver module of an Elliptic Curve and Galois Field HW Engine. If the target platform consists of such a dedicated HW accelerator then specific operations can be offloaded for additional latency improvements of the EPID operations. The design of such HW Engine is out of the scope of this work. It can be based on any of the existing HW Engines that supports short Weierstrass form of elliptic curves over 256-bit prime field (F_p) [8, 19, 22, 26]. The most significant optimization techniques used in our EPID software implementation are described as follows:

- EPID Prime Specific Reduction. We explored a reduction technique targeting the EPID prime modulus which reduces the reduction latency by 50% when compared to traditional Barrett reduction.

– Latency-Footprint Tread-off Exponentiations. EPID Sign/Verify operation consists of several exponentiation operations in G_1 and G_T which take most of the associated computation time. The lowest latency of those operations can be achieved by increasing the window size with several pre-computations. However, this increases the memory footprint for execution. We developed a footprint-and-latency balanced technique for computing Single as well as Multi-exponentiation operations in G_1 and G_T. The respective software functions are implemented in constant-time and with unified executions to prevent timing and simple power/EM vulnerabilities.
– Pre-computation of members pairing value and basic signature. There are four pairing values involved in EPID Sign which are computed on public and private-key components. Similar four pairings are involved in EPID Verify which are computed only on public-key components [12]. In the proposed EPID software stack, we compute them in an offline fashion and store them within the same public/private key data structure, thus reducing the significant latency overhead of Sign and Verify operations.
– Reduction of Verification Latency through revocation test followed by signature verification. In the EPID protocol, there are multiple revocation lists. For a signature verification, the Verifier typically establishes the proof for the signature and then validates that the signature/private-key have not been revoked. The later one is achieved by validating the revocation-proof embedded in the signature. In our investigation, we concluded that the revocation check requires a relatively lower latency when compared to the actual proof generation for the signature validation. Therefore, in our work, we first check that a given signature is valid across all revocation lists then we perform the actual signature validation thus saving a significant latency and compute energy for the verifier.

The paper is organized as follows. A brief introduction of EPID scheme is provided in Section 2. Our proposed software stack is described in Section 3. Section 4 provides our latency optimization techniques in the base-field and its elliptic curve group operations. Section 5 provides detailed descriptions of the extension fields and their execution in our EPID software. Optimal-ate pairing computation has been described in Section 6. EPID Join, Sign and Verify execution techniques are described in Section 7. The implementation results and comparison with state of the art implementations of the underlying functions are captured in Section 8. We conclude the paper in Section 9.

2 Background

As described in [14], the EPID scheme is comprised of four types of entities: an issuer, a revocation manager, users, and verifiers. The issuer can be the same entity as the revocation manager. An EPID scheme has the following four major functions:

– Setup: The issuer creates a group public key and a group issuing private key. The issuer publishes the group public key.

- Join: The issuer and the user cooperate to add the user to the group. The user obtains a membership private key unknown to the issuer.
- Proof of Membership: A prover interacts with a verifier to convince the verifier that he is a member of the group in good standing (i.e., not revoked). It has the following steps: (1) the prover sends a request to the verifier, (2) the verifier responds with a message m, (3) the prover generates a signature on m based on his membership private key, and (4) the verifier verifies the signature using the group public key.
- Revocation: The revocation manager puts a group member into one of the revocation lists. There are three types of revocations:

  1. Private-Key-Based Revocation: the revocation manager revokes a user's membership private key if it is discovered.
  2. Signature-Based Revocation: the revocation manager revokes a user based on a signature created by the user.
  3. Name-Base Revocation: a verifier forces users to use pseudonyms in name-base mode, and revokes individual pseudonyms.

The EPID scheme generates a proof-of-membership signature whose properties match common group signature [2, 6, 7, 16] properties: (1) unforgeability, i.e., only non-revoked group members are able to generate valid signatures, (2) anonymity, i.e., the verifier cannot identify the actual signer given a valid signature, and (3) unlinkablity, i.e., it is computationally infeasible to determine whether two different signatures were computed by the same group member. It was developed from the Direct Anonymous Attestation (DAA) scheme [9] and applied Camenisch-Lysyanskaya (CL) signature scheme [16]. For private-key based revocation, it uses verifier local revocation [7], i.e., the revocation check is done only at the verifier's side. For the other two types of revocation, EPID provides proof of knowledge protocols under the strong RSA assumption and the decisional Diffie-Hellman assumption to prove that a user's membership has not been revoked by the revocation list [14].

Many group signature schemes handle revocation by adding traceability where the revocation manager has the ability to open a signature and identify the actual signer. To revoke a user based on his signature, the revocation manager first finds out the user's private key or his identity, then put the user into the revocation list. However, EPID provides much higher privacy than traceability-based signature revocation. Traceability provides that a revocation manager can determine which user generated which signatures without any acknowledgment from the user that is being traced. This is not desirable from a privacy perspective. With EPID, if a user's membership private key has been revoked, i.e., placed in a revocation list, the user will know that he is revoked or being traced. If the user finds that he is not in the revocation list, then he is assured that nobody can trace him, including the issuer and the revocation manager, which is a major advantage of the EPID scheme.

2.1 EPID Parameters

The order of G_1, G_2, G_T is a 256-bit prime p. The characteristic of the finite field used in the elliptic curve is chosen as a 256-bit prime q. The parameter b defines the elliptic curve $y^2 = x^3 + b$. The parameter t is used to generate the prime $q = 36t^4 + 36t^3 + 24t^2 + 6t + 1$ and $p = 36t^4 + 36t^3 + 18t^2 + 6t + 1$. The variable neg is used to identify the sign of t and it is queried in the computation of the Optimal-ate pairing. The parameter β defines the extension field F_{q^2} over F_q. Similarly, the parameter ξ defines the extension field F_{q^6} over F_{q^2}.

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFCF0CD 46E5F25E EE71A49E} \\
&\quad \texttt{0CDC65FB 1299921A F62D536C D10B500D} \\
q &= \texttt{FFFFFFFF FFFCF0CD 46E5F25E EE71A49F} \\
&\quad \texttt{0CDC65FB 12980A82 D3292DDB AED33013} \\
b &= \texttt{00000000 00000000 00000000 00000000} \\
&\quad \texttt{00000000 00000000 00000000 00000003} \\
t &= \texttt{6882F5C0 30B0A801} \\
neg &= \texttt{1 (where neg is true if t is a negative value)} \\
\beta &= \texttt{-1} \\
&= \texttt{FFFFFFFF FFFCF0CD 46E5F25E EE71A49F} \\
&\quad \texttt{0CDC65FB 12980A82 D3292DDB AED33012} \\
\xi[0] &= \texttt{00000000 00000000 00000000 00000000} \\
&\quad \texttt{00000000 00000000 00000000 00000002} \\
\xi[1] &= \texttt{00000000 00000000 00000000 00000000} \\
&\quad \texttt{00000000 00000000 00000000 00000001}
\end{aligned}
$$

3 Optimized SW Architecture

Figure 1 depicts the hierarchical architecture and underlying functional layers of our embedded EPID Software-Hardware stack. Operations in the lower levels in shaded boxes can be performed by either hardware or software. There is a global flag in the embedded SW stack which indicates whether a hardware accelerator is available on the target platform. If no hardware accelerator is available then the entire hierarchy is implemented and executed in software. However, if the target platform contains a HW accelerator, then the software offloads those operations into the HW. Note that the underlying HW design is out of the scope of this work. However, in the following sub-section we provide how the current SW stack is configured and how it utilizes the underlying HW Engine.

In the proposed EPID SW stack, F_q is the finite field of order q. F_{q^2} is a degree-two finite field extension of F_q, of order q^2. F_{q^6} is a degree-three extension

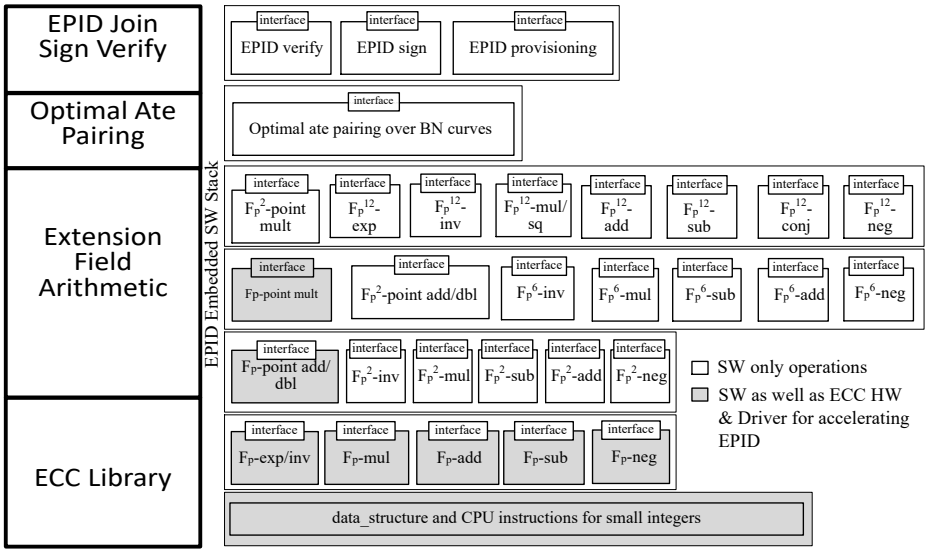

Fig. 1: Embedded SW stack for EPID.

of F_{q^2}, of order q^6. $F_{q^{12}}$ is a degree-two extension of F_{q^6}, of order q^{12}. Fq*Elem are data structures representing elements of their respective finite fields. EccPointFq holds affine (x,y pair) representations of points on the curve over F_q. EccPointJacobiFq holds Jacobi ($X : Y : Z$, corresponding to X/Z^2, Y/Z^3) representations of points on the curve over F_q. EccPointFq2 holds affine (x,y pair) representations of points on EPID's twisted curve $y^2 = x^3 + b/\xi$ over F_{q^2}. EccPointJacobiFq2 holds Jacobi representations of points on EPID's twisted curve over F_{q^2}. In general, Fq*Add computes the sum of two values, in its respective finite field; Fq*Sub computes the difference of two such values; Fq*Mult computes the product; Fq*Inv computes the multiplicative inverse of the input value; and Fq*Neg computes the additive inverse of the input value, in the finite field.

3.1 Utilization of HW Accelerator

An IoT device may optionally have an inbuilt HW accelerator for performing 256-bit Prime Field and Elliptic Curve operations. Figure 2 depicts a trivial architecture and interface of such a HW engine. It is comprised of an Elliptic Curve Cryptography (ECC) & F_p HW, the Interface and Bus Protocol Logic and a set of Addressable Registers. In the SoC environment, it is connected with an MCU which can perform read and write transactions in the ECC & F_p HW Engine's address space by Memory Mapped I/O (MMIO) instructions. Through a few writes into some of these addressable registers, the MCU can configure the HW accelerator for a specific operation. Similarly, the MCU can receive

the result of an operation from the HW accelerator by executing MMIO read instructions. We are not going into any further details about the HW accelerator in this paper and keeping major focus on the embedded EPID SW stack for IoT platforms. For further information about how to design and build such a HW engine we refer [8, 19, 22, 26] to the readers.

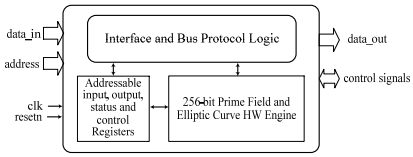

Fig. 2: The 256-bit Galois Field and Elliptic Curve HW Engine and Interface.

4 Latency Optimization in the Base Field

Galois Field over the 256-bit prime q is used as base field in the EPID scheme. We use BN curve group $E(F_q)$ over this base field with prime order p, which is used as the subgroup order in each of the three groups (G_1, G_2, G_T) used in EPID. Integer multiplication and reductions by p and q are the basic mathematical operations for computing each functional layers of the EPIS SW stack. The following subsection provides the optimized reduction technique for p and q that we explored and implemented in the current EPID SW stack.

4.1 EPID Specific Modulus Reduction

Our new EPID-prime reduction technique has been designed for the prime numbers used in EPID aiming at better efficiency on 32-bit processors. Algorithm 1 provides the detailed steps for computing the proposed reduction which is primarily based on Barrett reduction. It computes the reciprocal of the most significant 32 bits of the prime. Then it multiplies the Barrett multiplier (32-bit reciprocal) with the most significant 32 bits of the intermediate reduced result. It multiplies the most significant 32 bits of the last result with prime modulus. Then it subtracts this result from the most significant k+32 bits of the intermediate reduced result. Finally it left-shifts the intermediate result by 31 bits which is subsequently considered as the new intermediate reduced result. The above steps are repeated until the result is reduced to a k-bit number.

EPID has a prime q that is very close to 2^{256}, greater than $2^{256} - 2^{224}$. This lends itself to an efficient reduction method, targeted for 32-bit platforms, that reduces 31 bits at a time. We treat the highest-order 288 bits of a 512-bit number to be reduced like a 288-bit number n, and reduce it to a 257-bit number. Then we left shift the result by 31 bits and repeat the procedure until we ended up with the final 256-bit reduced result. For k the highest-order 32 bits of n, we know

Algorithm 1 EPID Modulus Reduction

Require: The 256-bit Modulus P, 512-bit integer M
Ensure: $N = M \bmod P$
  $i \leftarrow 8$
  while $i > 0$ **do**
    $R_{31:0} \leftarrow 2^{288}/P$
    $S_{31:0} \leftarrow (R * M_{511:224})_{543:512}$
    $T_{287:0} \leftarrow S_{31:0} * P_{255:0}$
    $U_{287:0} \leftarrow M_{511:224} - T_{287:0}$
    $M_{511:0} \leftarrow M_{511:0} << 31$
    $i \leftarrow i - 1$
  $M_{511:0} \leftarrow M_{511:0} >> 24$
  $S_{31:0} \leftarrow (R * M_{511:224})_{543:512}$
  $T_{287:0} \leftarrow S_{31:0} * P_{255:0}$
  $U_{287:0} \leftarrow M_{511:224} - T_{287:0}$
  $M_{511:0} \leftarrow M_{511:0} << 31$
  if $M_{511:256} < P_{255:0}$ **then**
    Return $M_{511:256}$
  else
    Return $M_{511:256} - P_{255:0}$

that $k*q <= n < k*q+2^{257}$ because $0 < n-k*q = n-k*2^{256}+(2^{256}-q)*k < (n \bmod 2^{256}) + 2^{224} * k \leq 2^{256} + 2^{256}$. This process can be repeated eight times, and then once more on the remaining eight bits, to reduce a 512-bit number to a 257-bit number, it computes $n' < 2^{256} + 2^{232}$. $n' - q$, and chooses either n' or $n' - q$ as the value of $n \bmod q$ based on whether $n' - q$ is at least, or less than, zero.

This method resembles Barrett reduction, but the multiplier is a power of 2, so the step of multiplying by the Barrett multiplier is eliminated; this improves the efficiency of this method without adding more memory cost.

5 Memory and Latency Trade-off in Extension Field

The EPID setup, join, sign verify operations require several extension field arithmetic operations. The optimized formula and implementation techniques of those mathematical operations are highlighted in the following subsections.

5.1 F_{q^2}, F_{q^6}, and $F_{q^{12}}$ Arithmetic

F_{q^2} arithmetic relies on $F_q[x]/(x^2 + 1)$, a polynomial irreducible over F_q. It is stored as a tuple, (a, b) of F_q elements representing a reduced F_q polynomial $a + bx$, where $x^2 = -1$. Addition and subtraction are pairwise addition in F_q, and multiplication is done in the same way as the fast complex multiplication: $(a, b)(a', b') = (a \cdot a' - b \cdot b', (a + b) \cdot (a' + b') - a \cdot a' - b \cdot b')$. Multiplicative inverse uses $(a', b') = (a, -b)/[(a, b)(a, -b)]$, where $(a, b)(a, -b)$ is a scalar in F_q.

F_{q^6} is $F_{q^2}[y]/(y^3 - \xi)$, where ξ is a non-square, non-cube element in F_{q^2}. Arithmetic is hard-coded to have as few multiplications as possible, as follows:

$$\begin{aligned}(a,b,c)\cdot(a',b',c') = \\ &(((b+c)(b'+c') - b\cdot b' - c\cdot c')\xi + a\cdot a',\\ &c\cdot c'\cdot\xi + (a+b)(a'+b') - a\cdot a' - b\cdot b',\\ &(a+c)(a'+c') - a\cdot a' - c\cdot c' + b\cdot b')\end{aligned}$$

The modular inverse is obtained by similar computation; we construct the conjugate that solves $(a,b,c)\cdot(a,b',c') = (k,0,0)$ for some k, and then divide by k:

$$\begin{aligned}(a',b',c') &= (a^2 - \xi b^2c^2, c^2\xi - a^2br, b^2 - a^2c^2)\\ (a',b',c')\cdot(a,b,c) &= (a'a + \xi b'c + \xi bc', 0, 0)\\ k &= a'a + \xi b'c + \xi bc'\\ (a,b,c)^{-1} &= (a'/k, b'/k, c'/k)\end{aligned}$$

$F_{q^{12}}$ is $F_{q^6}[z]/(z^2 - y)$, which is an extension field because ξ is not a square or a cube. Arithmetic is done in the same way F_{q^2} arithmetic is, with the exception that $(a,b)(a',b') = (aa' - ybb', (a+b)(a'+b') - aa' - bb')$, where y is the cube root of ξ, represented as the F_{q^6} element $(0,1,0)$.

5.2 $E(F_q)$ and $E'(F_{q^2})$ Group Operations

$E(F_q)$ is the elliptic curve over F_q subject to the equation $y^2 = x^3 + 3$, with order p. $E'(F_{q^2})$ is the sextic twist of this curve, over F_{q^2}, subject to the equation $y^2 = x^3 + 3/\xi$. ξ is picked so that the order of $E'(F_{q^2})$ is $p(2q-p)$. The formula for point addition on the curves is the same, because the Weierstrass curve addition formula does not depend on the constant.

Among the curves of form $y^2 = x^3 + ax + b$, setting $a = 0$ improves the performance of the curve because the addition formula includes an a term, but when $a = 0$, this term disappears. In the EPID software, we implemented the group operations on both the elliptic curves on Jacobian Projective Coordinates [17].

5.3 $F_{q^{12}}$ Multi-exponentiation

Multiexponentiation computes $A^aB^bC^cD^d$ for $A, B, C, D \in F_{q^{12}}$, $a, b, c, d \in F_p$. Typical approaches include computing four exponentiations by traditional binary exponentiation, running one loop in which an accumulator is conditionally multiplied by A, B, C, and D as necessary then squared, or computing a lookup table of [1, a, b, ab, c, ac, bc, abc, d, ad, bd, abd, cd, acd, bcd, $abcd$] and then multiplying the accumulator by the appropriate member of this table, as desired. The first approach avoids reusing code, but takes four times as long as the third approach; the second approach saves three squarings per iteration of the loop,

but squaring costs less than multiplication so it is desirable to also reduce the number of multiplications. Exponentiation with table lookup requires a lookup table with at least 11 entries, taking up 4 kB of RAM during runtime even if we assume a, b, c, and d are present as inputs; for simplicity, a table might be compiled of all 16 values, using 6 kB. Side-channel-resistant memory access requires polling all of these values every time a memory lookup is done, which makes its advantages less attractive. Instead, we compromised on a hybrid approach using two tables, one effectively containing $1, a, b, ab$ and one containing $1, c, d, cd$. This approach has the advantage of requiring only that we compute and store only ab and cd, with 2 entries instead of 11. We select from the table a, b, ab, and multiply by the accumulator, then select either the original or the new product; we again select from the table c, d, cd and multiply, then again select the accumulator or the new product as the new value of the accumulator.

Algorithm 2 EPID $F_{q^{12}}$ Multiexponentiation

Require: The $F_{q^{12}}$ elements A, B, C, and D, and F_p exponents a, b, c, and d
Ensure: $R = A^a B^b C^c D^d$
  $i \leftarrow 255$
  $R \leftarrow 1 \in F_{q^{12}}$
  $S \leftarrow 1 \in F_{q^{12}}$
  $ab \leftarrow a \times b$
  $cd \leftarrow c \times d$
  while $i >= 0$ **do**
    $S \leftarrow a$ if $a_i = 1$ else S
    $S \leftarrow b$ if $b_i = 1$ else S
    $S \leftarrow ab$ if $a_i = b_i = 1$ else S
    $S \leftarrow R \times S$
    $R \leftarrow S$ if $a_i \vee b_i = 1$ else R
    $S \leftarrow c$ if $c_i = 1$ else S
    $S \leftarrow d$ if $d_i = 1$ else S
    $S \leftarrow cd$ if $c_i = d_i = 1$ else S
    $S \leftarrow R \times S$
    $R \leftarrow S$ if $c_i \vee d_i = 1$ else R
    $i \leftarrow i - 1$
  Return $R = A^a B^b C^c D^d$

6 Optimal-ate pairing over BN Curve

Algorithm 3 computes the Optimal-ate pairing [31]. We chose a pairing friendly Barreto-Naehrig (BN) elliptic curve [4] $E : y^2 = x^3 + 2;\ z = -(2^{62} + 2^{55} + 1) < 0$ for efficiency, with the Optimal-ate pairing. The algorithm consists of two major parts - namely, *Miller's loop*, and *final exponentiation*. Several software and hardware implementations for the Optimal-ate pairing over BN

curves are reported in [5, 23, 25, 27, 29]. The EPID uses a negative t, because of which, after the Miller loop, we set accumulator $T = [-|s|]Q$, and the value of $f_{(s,Q)}(P)$ is raised to the power q^6 which is equivalent to f^{-1} as shown in [1]. The latter operation is computed by conjugation in $F_{q^{12}}$. Thereafter, the algorithm computes $g_{(sQ,\pi_q(Q))}(P)$ and $g_{(sQ+\pi_q(Q),-\pi_q^2(Q))}(P)$, which are multiplied into f.

Algorithm 3 Optimal-ate pairing on BN curve ($t < 0$)

Require: $P = (x_P, y_P) \in E(\mathbb{F}_q)$, $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^2})$, $s = 6t - 2 = \Sigma_{i=0}^{n-1} s_i 2^i$
Ensure: $a_{opt}(Q, P) \in \mathbb{F}_{p^{12}}$
$T \leftarrow (x_Q, y_Q, 1)$, $f \leftarrow 1$
for $i = n - 2$ downto 0 **do**
$g \leftarrow l_{(T,T)}(P)$, $T \leftarrow 2T$, $f \leftarrow f^2$, $f \leftarrow f \cdot g$
if $s_i = 1$ **then**
$g \leftarrow l_{(T,Q)}(P)$, $T \leftarrow T + Q$, $f \leftarrow f \cdot g$
$T \leftarrow -T$, $f \leftarrow f^{q^6}$
$Q_1 \leftarrow \pi_q(Q)$, $Q_2 \leftarrow -\pi_q^2(Q)$
$g \leftarrow l_{(T,Q_1)}(P)$, $T \leftarrow T + Q_1$, $f \leftarrow f \cdot g$
$g \leftarrow l_{(T,Q_2)}(P)$, $T \leftarrow T + Q_2$, $f \leftarrow f \cdot g$
$f \leftarrow \left(f^{q^6-1}\right)^{q^2+1}$
$f \leftarrow f^{(q^4-q^2+1)/p}$
return f

Algorithm 3 employs arithmetic in $\mathbb{F}_{q^{12}}$. High-performance arithmetic over extension fields is achieved through a tower of extensions using irreducible binomials as defined in Section 5. In the EPID protocol, we follow the towering $\mathbb{F}_{q^2} \rightarrow \mathbb{F}_{q^6} \rightarrow \mathbb{F}_{p^{12}}$ which makes the arithmetic (mainly squaring) required for this final exponentiation cheaper. The choice $q \equiv 3 \pmod 4$ accelerates arithmetic in $\mathbb{F}_{p^2}$, since $\beta = -1$ is a square-free element, and multiplications by β can be computed as simple subtractions [21]. The final exponentiation is computed in three major steps: $f \leftarrow f^{q^6-1}$, $f \leftarrow f^{q^2+1}$, and $f \leftarrow f^{(q^4-q^2+1)/p}$. The first two exponentiations are easily computed by using conjugate, inverse, Frobenius and multiplication operations in $F_{q^{12}}$. The last one is the most computationally intensive operation in the final exponentiation step. However, after raising f to the power $q^6 - 1$, it becomes a member of the cyclotomic subgroup in $F_{q^{12}}$ which allows underlying square and exponentiation to be easier than their naïve computation in $F_{q^{12}}$, which helps to speedup the last part of the final exponentiation.

7 Optimized EPID Functions

In order to make EPID feasible for IoT embedded devices with scarce computational capability, we developed techniques to minimize the necessary working memory and code size, besides targeting the implementation for 32-bits architec-

tures. Since it is unlikely that edge devices will play the issuer role, we focused on optimizing the operations required by members and verifiers.

7.1 EPID Setup and Join

In EPID, the issuer is responsible for setting up the EPID group for the purpose of device authentication. The EPID group corresponds to a group of trusted devices, as decided by the issuer. The group is created by generating a public/private keypair: the private key is a random F_p element that is not 0 nor 1, and the public key is a tuple $(\text{gid}, h1, h2, w)$, which consists of a group ID gid, random $E(F_q)$ elements $h1$ and $h2$, and $w = g_2^{\gamma}$, where g_2 is a parameter of EPID and the generator of the group $G_2 \subset E'(F_{q^2})$.

Once this group is created, the issuer must communicate with each member via a trusted channel, to assert that the member is the entity it claims to be. It is over this channel that the Join protocol occurs.

EPID Join occurs in several steps; first, the member obtains the group public key to which it is joining, $(\text{gid}, h_1, h_2, w)$. The member generates a private key f which remains a secret, and generates a membership request $F = h_1^f$ with proof of knowledge c, s, where $s = r + cf$ and $c = \text{HASH}(h_1^r, F)$ for some random $r \in F_p$, which is sent to the issuer. This membership request (F, c, s) is received by the issuer, who verifies that it is a valid membership request by checking that $H(h_1^s F^{-c}, F) = c = H(h_1^r, F)$. The issuer then computes a random x, and sends the member the membership credential (gid, A, x) where $x \in F_p$ is a random nonzero element and $A = (g_1 F)^{\frac{1}{x+\gamma}}$.

This credential is enough for the member to prove they have the private key f matching the membership credential A, x; to prove this, all that the member is required to do is show they possess A, x, and f such that $P(A, wg_2^x) = P(g_1 h_1^f, g_2) = P(g_1, g_2) + P(h_1^f, g_2) = P(g_1, g_2) + f \cdot P(h_1, g_2)$, which proves that $A = (g_1 F)^{\frac{1}{x+\gamma}}$. The pairing function gives EPID the flexibility to randomize and therefore obscure the identity of the member, by obscuring the identity-associated or secret key values of f, A and g_2^x, and by randomizing any values computed from them, while still proving that it possesses values which match in this way. These values can be anonymized, by randomly obscuring them, or pseudonymized, by basing one of the obscuring values on the hash of a known string. Pseudonymizing the strings links signatures signed with that pseudonym together, at the choice of the member, usually at the request of a verifier.

7.2 EPID Sign

The member prepares to sign messages with its private key by precomputing several pairing values; the signature itself is computed without running the expensive pairing function, but it still must computes several exponentiations in the field $F_{q^{12}}$. This requires us to optimize for $F_{q^{12}}$ multiexponentiation as described before, as well as elliptic curve operations.

Member precomputation, which occurs before any signature is signed, consists of the four Optimal-ate pairing computations $pm_1 = P(A, g_2)$, $pm_2 = P(h_1, g_2)$, $pm_3 = P(h_2, g_2)$, $pm_4 = P(h_2, w)$, where A belongs to the member's private key, h_1, h_2, w are part of the public key and g_2 is the generator of G_2.

EPID Sign must both prove possession of a private key and hide the identity of that private key. Signing a message m with private key A, x, f and group public key (gid, h_1, h_2, w) goes as follows:

A random name is produced, in the form of an elliptic curve point $B \in E(F_q)$. Random parameters a, r_x, r_f, r_a, r_b are generated to obscure the values of x, f, a, and b. Then the signature is computed as:

$$\begin{aligned}
b =& ax \\
K =& B^f \\
T =& Ah_2^a \\
R_1 =& B^{r_f} \\
R_2 =& pm_1^{-r_x} \times pm_2^{r_f} \times pm_3^{r_b - a \cdot r_x} \times pm_4^{ra} \\
c =& H(H(p||g_1||g_2||h_1||h_2||w||B||K||T||R_1||R_2)||m) \\
s_x =& rx + c \cdot x \\
s_f =& rf + c \cdot f \\
s_a =& ra + c \cdot a \\
s_b =& rb + c \cdot b
\end{aligned}$$

The final signature is $(B, K, T, c, s_x, s_f, s_a, s_b)$.

7.3 EPID Verify

Verification works backwards to reconstruct the values of R_1 and R_2 from c. The verifier also has four precomputed pairing values: $pv_1 = P(h_1, g_2)$, $pv_2 = P(h_2, g_2)$, $pv_3 = P(h_2, w)$, $pv_4 = P(g_1, g_2)$, where h_1, h_2, w are part of the public key and g_1, g_2 are the generator of G_1, G_2 respectively.

$$\begin{aligned}
R_1 =& K^{-c} \cdot B^{sf} \\
R_2 =& P(T, g_2^{-s_x} w^{-c}) \times pv_1^{sf} \times pv_1^{sb} \times pv_1^{sa} \times pv_1^{c}
\end{aligned}$$

The verifier verifies that R_1 and R_2 are correct by recomputing the hash that the signer produced, proving that R1 and R2 are correct, which therefore proves that the relationship between $P(B, g_2)$ and $P(K, g_2)$ is the same exponent as exists between $P(h_1, g_2)$ and $P(h1^{\frac{f}{\gamma+x}}, wg_2^x)$, which is to say the exponent f. γ is hidden from the signer by the value of x, which is hidden from the verifier with r_x. The other values exist to hide the values of the variables in question.

The verifier has to also ensure that the member is not on any of the revocation lists. The verifier does this in three ways.

Private key revocation: The private key revocation list is a list of private keys f that have been exposed and subsequently revoked. Because $K = B^f$ in the signature provided, the verifier has only to check that $K \neq B^{f'}$ for each f' in the list. If the two were to match, the member key has been revoked.

Group revocation: If the group has been revoked, the verifier rejects all signatures signed under that group. This only requires checking the gid. Verifier-based revocation: The verifier can request that signers sign using $B = H(\text{pseudonym})$. The signer then produces signatures with a unique, identifiable K, which is constant only when signing messages under that pseudonym. The verifier has the option to restrict or block certain members by adding this K to the list and rejecting signatures whose B and K match this list.

Signature revocation: This revocation is interactive; the user must submit proofs that it is not on the list of revoked signatures. For each revoked signature B', K', the member chooses random nonzero F_p element μ and $nu = -f \cdot \mu$, and random nonzero F_p elements r_μ, r_ν. Then the member computes:

$$\begin{aligned} T &= K'^{\mu} B'^{\nu} \\ R_1 &= K^{r_\mu} B^{r_\nu} \\ R_2 &= K'^{r_\mu} B'^{r_\nu} \\ c &= H(p||g1||B||K||B'||K'||T||R_1||R_2||m) \\ s_\mu &= r_\mu + c \cdot \mu \\ s_\nu &= r_\nu + c \cdot \nu \end{aligned}$$

$\sigma = (T, c, s_\mu, s_\nu)$; T is essentially $K^{(f'-f)\mu}$, but by the signer proving it knows μ and ν, and that $\nu = -\mu \cdot f$, it pre-commits to the exponents being the same. If $\nu \neq -\mu \cdot f$, then $R_1 \neq K^{s_\mu} B^{s_\nu}$; the member cannot force a particular value of c, so this amounts to a proof of that relationship. The same is true with respect to the value of T and the relationship between B' and K', which also proves that the signature containing B' and K' was signed by a different member. The verifier computes $R_1 = K^{s_\mu} B^{s_\nu} T^{-c}, R_2 = K^{s_\mu} B^{s_\nu}$, and recomputes c to make sure the value is correct.

8 Footprint and Performance

The complete embedded EPID software stack was developed in C and designed targeting 32-bit microcontrollers as [24] and [28] demonstrated runtime data that discourage the use of any smaller (8-bit and 16-bit) microcontrollers for computing public key operations. The object code of the signer/verifier/issuer library is 22k bytes, compiled with gcc using the compilation flag: -Os, targeted for a representative IoT platform based on a low-power 32-bit Intel Quark microcontroller with 80k bytes internal SRAM. Using static code analysis, we calculated that the sign/verify operations consume about 10k bytes in the memory stack. Table 1 provides the latency of underlying operations of our EPID implementation on Intel® Curie® IoT platform.

Table 1: Latency of Galois Field and Elliptic Curve Operations on Intel® Curie®

Operation*	**F_q-M**	**F_{q^2}-M**	**$F_{q^{12}}$-M**	**$F_{q^{12}}$-ME**	**E(F_q)-SM**	**E(F_{q^2})-SM**
Clock Cycles	8,900	27,998	553,459	437,623,937	54,798,533	175,300,779

* **M**: Multiplication, **ME**: Multi-exponentiation, **SM**: Scalar multiplication.

The Optimal-ate pairing is one of the most expensive operations used by EPID. There are four such pairing values that are involved in EPID signing operation. Each member precomputes these values and uses them for signing multiple messages with the same key. Table 2 provides a performance comparison of our software implementation of the Optimal-ate pairing, running on Intel®'s Curie® platform with some of the existing implementations in different platforms. The Optimal-ate SW running on Intel Core processors are significantly faster than microcontrollers due to two main reasons: 64-bit instructions and the availability of high volume cache memory.

Table 2: Latency (Clock Cycles) of Optimal-ate pairing on 256-bit BN curves

Work, Year	**Platform**	**Latency (Clock Cycles)**
[27], 2010	Intel Core 2 Quad Q6600 (64-bit)	4,470,408
[5], 2010	Intel Core i7 (64-bit)	2,330,000
[1], 2011	Intel Core i5 (64-bit)	1,688,000
[30], 2014	ARM Cortex-M0 + HW module	47,643,000
Ours, 2018	Intel Curie (32-bit MCU)	188,053,871

Table 3 provides the performance of the full EPID software stack on Intel Curie platform. On Intel Curie platform, one EPID sign takes 572,902,590 clock cycles or 17.9s when it runs at 32MHz clock. Similarly, our experiment shows each Verify takes 1,020,079,132 clock cycles or 31.9s at the same clock frequency. It also provides additional results on Intel Core i5 with 32-bit mode. It is observed that there are significant latency difference between two 32-bit platforms which is mainly due to the availability of high-speed cache memory in the Core i5 processor whereas Curie is based on 32-bit Intel Quark microcontroller with 80kB low-speed SRAM.

Table 3: Performance of EPID SW stack on Intel Curie

	Intel Curie (32-bit) (Clock Cycles)	**Intel Core i5 (32-bit) (Clock Cycles)**
EPID Sign	572,902,590	194,199,881
EPID Verify	1,020,079,132	347,854,538

As we discussed before, the performance of the EPID scheme can be improved further by utilize the underlying ECC & F_p HW Engine and implementing/compiling our proposed software with a specific knob. In typical such hardware engines [8, 19, 22, 26] each modulus multiplication can be computed in as low as 100's of clock cycles compared to $8,900$ clock cycles on a Intel Curie platform. Therefore, our proposed EPID software stack in a suitable IoT platform can perform one EPID Sign or anonymous attestation in less than $1s$.

9 Conclusion

IoT devices are used in a wide range of applications that deal with privacy sensitive data. Enhanced Privacy ID (EPID) is an industry standard that provides anonymous attestation. EPID Sign/Verify operations involve large number arithmetic, and are computationally and memory intensive. This makes it challenging for deploying EPID in resource constrained IoT devices. In this paper, we have proposed a design for small footprint anonymous attestation based on EPID, and built a prototype to demonstrate its feasibility for IoT. The EPID SW stack has a small object code footprint of 22kB. Results from our prototype on a 32-bit microcontroller show that EPID Sign has latency of $572,902,590$ clock cycles or 17.9s at 32MHz.

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. pp. 48–68 (2011). https://doi.org/10.1007/978-3-642-20465-4_5, `https://doi.org/10.1007/978-3-642-20465-4_5`
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 255–270. Springer (2000). https://doi.org/10.1007/3-540-44598-6_16, `https://doi.org/10.1007/3-540-44598-6_16`
3. Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.): Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004. ACM (2004)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S.E. (eds.) Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3897, pp. 319–331. Springer (2005). https://doi.org/10.1007/11693383_22, `https://doi.org/10.1007/11693383_22`
5. Beuchat, J., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal

ate pairing over barreto-naehrig curves. In: Pairing-Based Cryptography - Pairing 2010 - 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings. pp. 21–39 (2010). https://doi.org/10.1007/978-3-642-17455-1_2, https://doi.org/10.1007/978-3-642-17455-1_2

6. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M.K. (ed.) Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3152, pp. 41–55. Springer (2004). https://doi.org/10.1007/978-3-540-28628-8_3, https://doi.org/10.1007/978-3-540-28628-8_3
7. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri et al. [3], pp. 168–177. https://doi.org/10.1145/1030083.1030106, http://doi.acm.org/10.1145/1030083.1030106
8. Bosmans, J., Roy, S.S., Järvinen, K., Verbauwhede, I.: A tiny coprocessor for elliptic curve cryptography over the 256-bit NIST prime field. In: 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, January 4-8, 2016. pp. 523–528. IEEE Computer Society (2016). https://doi.org/10.1109/VLSID.2016.82, https://doi.org/10.1109/VLSID.2016.82
9. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri et al. [3], pp. 132–145. https://doi.org/10.1145/1030083.1030103, http://doi.acm.org/10.1145/1030083.1030103
10. Brickell, E., Chen, L., Li, J.: A new direct anonymous attestation scheme from bilinear maps. In: Lipp, P., Sadeghi, A., Koch, K. (eds.) Trusted Computing - Challenges and Applications, First International Conference on Trusted Computing and Trust in Information Technologies, Trust 2008, Villach, Austria, March 11-12, 2008, Proceedings. Lecture Notes in Computer Science, vol. 4968, pp. 166–178. Springer (2008). https://doi.org/10.1007/978-3-540-68979-9_13, https://doi.org/10.1007/978-3-540-68979-9_13
11. Brickell, E., Li, J.: Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities. In: Ning, P., Yu, T. (eds.) Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, Alexandria, VA, USA, October 29, 2007. pp. 21–30. ACM (2007). https://doi.org/10.1145/1314333.1314337, http://doi.acm.org/10.1145/1314333.1314337
12. Brickell, E., Li, J.: A pairing-based DAA scheme further reducing TPM resources. In: Acquisti, A., Smith, S.W., Sadeghi, A. (eds.) Trust and Trustworthy Computing, Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6101, pp. 181–195. Springer (2010). https://doi.org/10.1007/978-3-642-13869-0_12, https://doi.org/10.1007/978-3-642-13869-0_12
13. Brickell, E., Li, J.: Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. IJIPSI **1**(1), 3–33 (2011). https://doi.org/10.1504/IJIPSI.2011.043729, https://doi.org/10.1504/IJIPSI.2011.043729
14. Brickell, E., Li, J.: Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. IEEE Trans. Dependable Sec. Comput. **9**(3), 345–360 (2012). https://doi.org/10.1109/TDSC.2011.63, https://doi.org/10.1109/TDSC.2011.63

15. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. Lecture Notes in Computer Science, vol. 2045, pp. 93–118. Springer (2001). https://doi.org/10.1007/3-540-44987-6_7, https://doi.org/10.1007/3-540-44987-6_7
16. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers. Lecture Notes in Computer Science, vol. 2576, pp. 268–289. Springer (2002). https://doi.org/10.1007/3-540-36413-7_20, https://doi.org/10.1007/3-540-36413-7_20
17. D Hankerson, A Menezes, S.V.: Guide to elliptic curve cryptography. Springer Science & Business Media (2006)
18. (editor), P.J.L., (co editor), S.M.: Text for iso/iec fdis 20009-2 — information technology — security techniques — anonymous entity authentication — part 2: Mechanisms based on signatures using a group public key. In: ISO/IEC JTC 1/SC 27 N12580 (2013)
19. Feng, X., Li, S.: A high-speed and spa-resistant implementation of ECC point multiplication over gf(p). In: 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, August 1-4, 2017. pp. 255–260. IEEE (2017). https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.245, https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.245
20. Furukawa, J., Imai, H.: An efficient group signature scheme from bilinear maps. IEICE Transactions **89-A**(5), 1328–1338 (2006). https://doi.org/10.1093/ietfec/e89-a.5.1328, https://doi.org/10.1093/ietfec/e89-a.5.1328
21. Geovandro, C.C.F.P., Jr., M.A.S., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly BN elliptic curves. Journal of Systems and Software **84**(8), 1319–1326 (2011). https://doi.org/10.1016/j.jss.2011.03.083, https://doi.org/10.1016/j.jss.2011.03.083
22. Ghosh, S., Mukhopadhyay, D., Chowdhury, D.R.: Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable gf(p) arithmetic unit. IEEE Trans. on Circuits and Systems **58-I**(8), 1798–1812 (2011). https://doi.org/10.1109/TCSI.2010.2103190, https://doi.org/10.1109/TCSI.2010.2103190
23. Ghosh, S., Verbauwhede, I., Chowdhury, D.R.: Core based architecture to speed up optimal ate pairing on FPGA platform. In: Abdalla, M., Lange, T. (eds.) Pairing-Based Cryptography - Pairing 2012 - 5th International Conference, Cologne, Germany, May 16-18, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7708, pp. 141–159. Springer (2012). https://doi.org/10.1007/978-3-642-36334-4_9, https://doi.org/10.1007/978-3-642-36334-4_9
24. Gouvêa, C.P.L., Oliveira, L.B., López, J.: Efficient software implementation of public-key cryptography on sensor networks using the msp430x microcontroller. Journal of Cryptographic Engineering **2**(1), 19–29 (May 2012). https://doi.org/10.1007/s13389-012-0029-z, https://doi.org/10.1007/s13389-012-0029-z
25. Gouvêa, C.P.L., López, J.: Software implementation of pairing-based cryptography on sensor networks using the MSP430 microcontroller. In: Roy, B.K., Sendrier, N. (eds.) Progress in Cryptology - INDOCRYPT 2009, 10th Inter-

national Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5922, pp. 248–262. Springer (2009). https://doi.org/10.1007/978-3-642-10628-6_17, https://doi.org/10.1007/978-3-642-10628-6_17

26. Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over fp. In: Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems. pp. 48–64. CHES'10, Springer-Verlag, Berlin, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1881511.1881517
27. Naehrig, M., Niederhagen, R., Schwabe, P.: New software speed records for cryptographic pairings. In: Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings. pp. 109–123 (2010). https://doi.org/10.1007/978-3-642-14712-8_7, https://doi.org/10.1007/978-3-642-14712-8_7
28. Szczechowiak, P., Kargl, A., Scott, M., Collier, M.: On the application of pairing based cryptography to wireless sensor networks. In: Proceedings of the Second ACM Conference on Wireless Network Security. pp. 1–12. WiSec '09, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1514274.1514276, http://doi.acm.org/10.1145/1514274.1514276
29. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Software implementation of binary elliptic curves: Impact of the carry-less multiplier on scalar multiplication. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. pp. 108–123 (2011). https://doi.org/10.1007/978-3-642-23951-9_8, https://doi.org/10.1007/978-3-642-23951-9_8
30. Unterluggauer, T., Wenger, E.: Efficient pairings and ecc for embedded systems. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014. pp. 298–315. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
31. Vercauteren, F.: Optimal pairings. IEEE Trans. Information Theory **56**(1), 455–461 (2010). https://doi.org/10.1109/TIT.2009.2034881, https://doi.org/10.1109/TIT.2009.2034881