Performance Analysis of Identity-based Signatures

Project Report — PR1

Tobias Markmann
tobias.markmann@haw-hamburg.de

August 13, 2014

This report analyzes the viability of identity-based signature (IBS) for providing authentication to the Internet of Things (IoT). A large part of the IoT are constrained devices with limited power, performance and memory. We provide a theoretical comparison and a practical evaluation based on benchmarks for there IBS. Our test subjects are the RSA based SH–IBS scheme, the elliptic curve cryptography (ECC) based vBNN–IBS and TSO-IBS which uses pairing-based cryptography (PBC). The schemes are compared at different security levels including standard and modern elliptic curves. The schemes are implemented using Charm and the RELIC toolkit. Our Charm implementation serves as initial proof of concept to validate and understand the signatures following with an implementation in C/C++ using the RELIC toolkit, a cryptographic C library for constrained devices. Finally, the C/C++ implementations are benchmarked. Our benchmarks are performed on a 64 bit Core i7 desktop platform and on a 32 bit Raspberry Pi. We show that ECC based IBS are a good option for constrained devices due to their better overall scalability and their platform specific advantages compared to RSA based SH-IBS.

Contents

1 Introduction

Wireless sensor networks (WSNs) and the Internet of Things (IoT) are major examples for networks of constrained devices. Especially the IoT is a permanently rising market and research area. Some form of authentication needs to be implemented to secure these networks against abusive users and other attackers [1].

Traditionally, computationally restricted computer networks like WSNs employed symmetric cryptography to provide authentication in communication. This was mainly due to very low computational complexity of symmetric cryptography. However it leads to more complex key distribution schemes. In addition, authenticated multicast protocols based on symmetric cryptography come with more protocol complexity in the event of group membership changes. One example for this is the µTESLA [2] protocol.

Asymmetric cryptography simplifies key management and allows to design less complex group communication protocols. A decade ago, asymmetric cryptography was deemed too expensive for constrained devices. Advances in low-power embedded hardware and in research of more efficient cryptographic primitives make asymmetric cryptography an interesting option for security solutions for constrained devices.

Classic asymmetric cryptosystems like public-key infrastructure (PKI) use certificates to bind identity information (email addresses or domain names) to keys. This means the certificate is required alongside the actual signature to verify it properly. These certificates need to be transferred and processed in addition to the signature and thereby increase the overall transmission cost.

In contrast, identity-based signatures (IBSs) as proposed by Shamir [3], have the identity binding implicitly integrated in the signatures generated by the users of the system. This allows signatures to be verified solely based on identity information, like IP addresses or MAC addresses. Addresses are easily available in packets of communication protocols and the public key for verification of the signature can be deduced from them.

Identity-based cryptography (IBC) works well to enable end-to-end security in private networks with a central base station or gateway. An example for this are WSNs. WSNs typically have a central base station which collects all the data and manages the general access to the network. Since this base station is trusted by all the nodes within the WSN it is an ideal candidate for the trusted authority (TA) in an IBC system. The TA generates all private keys for the users and the system is thereby subject to *key escrow*. The property of *key escrow* is less of a problem in a singly owned system than it would be in a multi-party distributed communication network, like the IoT.

In this work, three different IBSs schemes are implemented and evaluated. The original ID-based signature SH–IBS, an elliptic curve based IBS vBNN-IBS and an ID-based signature using bilinear pairings, TSO-IBS. All schemes are implemented in Python using Charm [4] for initial conceptual evaluation and in C++ using the RELIC toolkit [5], a cryptographic library for resource-constrained computers. The C/C++ implementations of the schemes are compared in benchmarks. The benchmark is carried out on both, a recent desktop platform (64 bit Intel Core i7) and a smaller, less powered, embedded platform (32 bit ARM, Raspberry Pi).

This report is structured as follows. Section 2 provides a brief introduction to related projects including a theoretical comparison of IBS, a security analysis of established and new elliptic curves and a low power comparison of RSA and elliptic curve cryptography (ECC) on 8-bit CPUs. In Section 3, IBC is introduced and its key management characteristics are compared to those of traditional PKI. This section also describes the basics of elliptic curve cryptography, its mathematical background and concludes with a brief description of pairing-based cryptography (PBC). In Section 4, the three signature schemes, SH–IBS, vBNN–IBS and TSO–IBS, based on RSA, elliptic curves and bilinear parings respectively, are described, compared and the implementation setup is specified. The next section presents the results of the executed benchmark. Finally, in Section 6, the results are interpreted in the light of the current state of the IoT. An analysis and outlook into future possible work is given.

1.1 Nomenclature

The mathematical notation used in this document is listed in Table 1.

Notation	Description
$G, G_1, G_T, ...$	*group*
p	*prime number*
$\mathbb{F}_p$, $\mathbf{GF}(p)$, $\mathbb{Z}/p\mathbb{Z}$	*finite field* of prime order with p elements
$\mathbb{F}_{p^n}$, $\mathbf{GF}(p^n)$, $\mathbb{Z}/p^n\mathbb{Z}$	*finite field* of prime power order with p^n elements

Table 1: Clarification of mathematical notation used in this report.

The nomenclature used for variables in cryptographic protocols is shown in Table 2.

Variable	Description
ID_A	identity of user A
$ID_{A_{key}}$	ID-based private key of user A
msk	*master secret key* of a TA
mpk	*master public key* of a TA
σ	signature

Table 2: Variable nomenclature used in this report.

The common notation for operations on buffers is shown in Table 3 at a glance.

Notation	Description
$a \oplus b$	XOR two buffers
$_n\|a\|$	first n bits of buffer a
$\|a\|_n$	last n bits of buffer a
$a\|\|b$	concatenate two buffers a and b
$\|q\|$	bit length of the binary representation of integer q

Table 3: Clarification of notation of buffer operations in this work.

2 Related Work

Kiltz and Neven [6] theoretically analyzed and compared various ID-based signature schemes with regard to signature size, computational complexity and security strength. This comparison provides a good high-level overview on the computational and storage complexity of the various schemes. This is specified in terms of group operations for computation complexity and number of group elements for space complexity. However, there is no practical evaluation via implementation and benchmark. In addition, it is an analysis solely in the area of identity-based signature (IBS) without showing complexity relative to classic asymmetric signatures.

SafeCurves [7] is an ongoing project by Bernstein and Lange on providing an overview of popular choices of elliptic curves. The project compares the curves with regards to their resilience against commonly known attacks and weaknesses in the area of elliptic curve cryptography (ECC). The authors do not provide a performance comparison between elliptic curves. Some curves for the benchmark carried out in our project have been specifically chosen from this list to extend test candidates beyond the standardized curves.

Gura, Patel, Wander, *et al.* [8] compare standard ECC against RSA signatures on very low power embedded 8-bit CPUs. They analyze and implement the cryptographic algorithms in assembly code. 160 bit ECC point multiplication is shown to outperform 1024 bit RSA private-key operation. 160 bit ECC point multiplication is twice as fast as compared to 1024 bit RSA public key operation. While this comparison is limited to classic asymmetric signatures and executed on extremely low power embedded CPUs, it provides an interesting analysis helping to set our own benchmarks results in perspective, that are obtained on more powerful hardware.

3 Background

This section serves as introduction to the background knowledge required for the following description of cryptographic signature schemes. Starting out with cryptographic background of identity-based cryptography (IBC), mathematical fundamentals required for modern asymmetric cryptographic protocols are explained. This section completes with basics of elliptic curve cryptography (ECC) and pairing-based cryptography (PBC).

3.1 Identity-based Cryptography

Identity-based cryptography is a form of asymmetric cryptography—also known as public-key cryptography (PKC)—that was first suggested by Shamir in 1985 [3]. It simplifies the key distribution problem in cryptographic systems compared to symmetric cryptography. For identity-based signatures (IBSs) an arbitrary string which uniquely identifies an entity can be used as public key (ID). The ID of an entity and publicly known general parameters are enough to verify signatures generated by that entity.

Since this work concentrates on authenticating communication in networks of devices with low computing power, we will focus on ID-based signature schemes.

Definition 3.1. An IBS is defined using four functions:

$$\begin{aligned}
Setup(sec\_level) &\longrightarrow & (msk, mpk, pub\_params) \quad (1)\\
KeyExtract(msk, mpk, pub\_params, ID) &\longrightarrow & (ID_{key}) \quad (2)\\
Sign(mpk, pub\_params, ID_{key}, m) &\longrightarrow & (\sigma) \quad (3)\\
Verify(mpk, pub\_params, ID, m, \sigma) &\longrightarrow & true/false \quad (4)
\end{aligned}$$

IBSs according to Definition 3.1 can be used in the following way. Initially the system is set up according to the desired security level. For each user, a private key needs to be extracted using the $KeyExtract$-function. Using the private key, users can sign messages producing signatures (σ) of messages, which are verifiable using solely public parameters and the public key derived from the ID.

In traditional PKC systems, the private/public key pair is generated by each user. However, to use asymmetric cryptography for securing communication, the receiver needs to know the public key of the sender. The binding between sender and her public key needs to be secure in a way that an attacker cannot spoof a different public key and pretend to be someone else. This can be achieved in various ways.
One way is to statically pre-distribute the public key/identity mapping to all communication partners, which comes with a huge overall storage cost. A more practical approach is to send the public key/identity binding over the wire in combination with a signature from a third party covering the identity binding. This way, all participants only need to have the public key of the third party to verify the public key/identity binding. While this increases packet sizes, it hugely reduces storage overheads for public keys on user devices. This is essentially the way the public key/identity binding is done in the public-key infrastructure (PKI) for the world wide web (WWW). In the practical deployment there is more than one third party, known as certificate authority (CA), that can sign a public key/identity binding and there is a hierarchy of CAs to ease certificate management.

A different approach to the key management problem is to use IBC. IBC can reduce package overhead and further simplify the key management. Here the public key can be deduced from identity information, commonly an email address or Internet protocol (IP) address depending on the scenario. In contrast to classical PKC, the private keys need to be generated by the commonly trusted third party, called trusted authority (TA), key generation center (KGC) or key-generating server (KGS) in IBC. This outsourcing of private key generation to a trusted party is called *key escrow* and is a critical property of simple IBC systems.

A third party knowing the private keys of other users is able to decrypt all messages in an identity-based encryption (IBE) scheme and is able to forge signatures for any message and any user in an IBS scheme. Thus IBS cannot offer real non-repudiation. In addition, this third party is an attractive target for attacks because access to it discloses all private information of the crypto system.

There are proposals to mitigate the *key escrow* problem in IBC systems. Boneh and Franklin [9] suggest distributing the KGC over multiple servers where each server only holds part of the master secret key. To gain access to the full master secret key multiple servers need to collude. Distributed KGCs are widely discussed and proven secure in [10]. Another proposal by Al-Riyami and Paterson is certificateless public-key cryptography (CL-PKC) [11]. In CL-PKC the final private key is generated by the user based on secret information from the KGC and secret information of the user. This way the KGC cannot forge signatures and the system is free of *key escrow*. However, the system is not ID-based anymore, because public keys are no longer derivable from IDs.

3.2 Asymmetric Key Management

Employing asymmetric cryptography in real applications comes with essential auxiliary tasks as part of the key management. To provide a good level of security to all members of a communication system, developers have to bow to the inevitable and prepare their systems for incidents like key exposure and key renewal. Key exposure usually requires key renewal, since the now publicly known private key could be used to impersonate a legitimate user.

There are various ways a user's private key could be exposed. This can happen through human error in manual processes and more critically, due to bugs in security relevant protocol implementations exposed to public networks, as it has been shown recently with the Heartbleed bug in OpenSSL [12]. After possible exposure of a private key to the public there is only one correct process; the certificate associated with the exposed key needs to be revoked to limit possible damage and a new certificate with new private/public key pair needs to be obtained.

3.2.1 Traditional Public-key Infrastructure

In a PKI as used by the WWW, users can revoke their certificate at the issuing party, the CA. However, the task of actively checking certificates for their current revocation state is left to the clients. Basically there are two different kind of approaches to this problem:

1. *offline / asynchronous*: during the verification step each user checks the certificate against a list of keys that have been revoked, the so called certificate revocation list (CRL). This list is issued by a CA and signed using their private key. CRLs can be downloaded on demand or pushed to the users on a regular basis. The handling of CRLs in the PKI is described in more detail in [13].
2. *online / synchronous*: for verification, the user asks a predefined server about the current revocation state of a specific certificate to which the server responds with a signed reply containing the current revocation state. This protocol for the PKI is called Online Certificate Status Protocol (OCSP) [14] and allows a secure on-demand attestation on the state of revocation of a certificate. The OCSP server is provided by the CA that issued the certificate.

The offline approach comes with a scalability issue. The list of revoked certificates only ever increases and has to include all revoked certificates that would otherwise still be valid. In addition, the CRLs need to be updated at all clients at a regular bases to correctly detect revoked certificates.

In contrast, OCSP does not require updating huge lists of certificates. However, it requires that all CAs have an OCSP server running which will reply to the requests of clients checking the revocation state of certificates. In a setup which prioritizes security over usability, a certificate would be considered revoked if an OCSP server is not reachable. Thus, the OCSP server introduces a single point of failure which may be under heavy load considering each validation of a certificate requires a request to an OCSP server.

To reduce load on the OCSP server, an optimization has been proposed. OCSP stapling [15] allows the verifier of a certificate to immediately check the revocation state without further contacting an OCSP server. This is possible because the communication partner already requested an OCSP response from the CA and attached it to the message to be verified.

3.2.2 ID-based Cryptography

While IBC eases key distribution compared to traditional PKI, the problems of key management are of at least similar complexity. Especially the inherent implicit binding of public key and identity in IBC makes it hard to revoke keys for users. Simply adding the identity to a revocation list would prevent the user of that identity from ever sending signed messages again. Early systems using IBC avoided the classic approach of revocation altogether and instead went with automatic key renewal [9]. Here Boneh and Franklin proposed to add time related information to the identity before deriving the associated public key from it, e.g. `identity + year`. In this way users are required to get a new private key from the TA each year and the TA just stops handing out private keys to revoked users. However, adding time related information requires all users to fetch a new private key from the TA in the common time frame.

Revoking public keys in an IBC system equals revoking the associated identity. If however the identity is hard to change (like static IP addresses), the identity string needs to be extended to still allow revocation and rekeying. Extending the identity string with additional information is nontrivial. Identities in IBC systems need to be easy to predict to allow easy verification. Adding a rough timestamp as proposed by Boneh and Franklin still allows easy verification considering loosely synchronized clocks between the users. However, adding hard to predict data, e.g. issue numbers, to identities complicates the verification process as information about the current valid issue number needs to obtained out-of-band [16, p. 64].

Boldyreva, Goyal, and Kumar [17] proposed an IBE system with improved revocation handling, providing logarithmic scaling for maintenance work for all users of the system as compared to linear within a revocation time frame. However, it heavily uses PBC and thus comes with great computational complexity.

Considering the two major environments where constraint devices are in wide use—the Internet of Things (IoT) and wireless sensor networks (WSNs)—the key management properties are of different relevance in each environment. While required continuous key updates within the revocation timeframe in an IBC setup might well work in small WSNs with up to 1000 nodes, in a system at the scale of the IoT this could become a greater problem. The flexible key management in traditional PKI setups on the other hand allow for various different ways of providing revocation information to the users of the system and is easily distributable.

3.3 Mathematical Background of Groups, Rings and Fields

In this section, the basic mathematical building blocks are described for quick reference in this work. This covers groups, rings, and finite fields, the requirements for crypto systems based on numbers modulo prime and ECC.

Shoup [18, p. 126] defines a *group* as the following:

Definition 3.2. An *abelian group* is a set G together with a binary operation $\cdot$ on G such that:

1. for all $a, b, c \in G$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, (i.e. $\cdot$ is associative),
2. there exists $e \in G$ (called the *identity element*) such thar for all $a \in G$, $a \cdot e = a = e \cdot a$,
3. for all $a \in G$ there exists $a' \in G$ (called the *inverse of* a) such that $a \cdot a' = e = a' \cdot a$;
4. for all $a, b \in G$, $a \cdot b = b \cdot a$ (i.e., $\cdot$ is commutative).

He also defines[18, p. 166] a *ring* as:

Definition 3.3. A *commutative ring with unity* is a set R together with addition and multiplication operations on R, such that:

1. the set R under addition forms an abelian group, and we denote the additve identity by 0_R,

2. multiplication is associative; that is, for all $a, b, c \in R$, we have $a(bc) = (ab)c$,
3. multiplication distributes over addition; that is, for all $a, b, c \in R$, we have $a(b+c) = ab + ac$ and $(b + c)a = ba + ca$,
4. there exists a multiplicative identity; that is, there exists an element $1_R \in R$, such that $1_R \cdot a = a = a \cdot 1_R$ for all $a \in R$,
5. multiplication is commutative; that is, for all $a, b \in R$, we have $ab = ba$.

Definition 3.4. The *characteristic* of a ring R, is defined as the smallest number of times one has to add the multiplicative identity, 1, with itself to get the additive identity, 0. In the case the additive identity is never reached, the *characteristic* is defined as 0. It is often written as $char(R)$.

Menezes, Oorschot, and Vanstone define *fields* [19, p. 77] and *finite files* [19, p. 80-81] in the following way:

Definition 3.5. A *field* is a commutative ring in which all non-zero elements have multiplicative inverses.

Definition 3.6. A *finite field* is a field F which contains a finite number of elements. The *order* of F is the number of elements in F.

Finite fields are also known as *Galois fields* (**GF**) [20, p. 31] and main examples for *finite fields* are:

1. Prime fields: $\mathbb{F}_p$ or $\mathbf{GF}(p)$ where p is prime
2. Extensions of fields: $\mathbb{F}_{p^n}$ or $\mathbf{GF}(p^n)$ where p is prime and n is a non-zero positive integer

The characteristic of the field is equal to the prime p: $char(\mathbb{F}_p) = char(\mathbb{F}_{p^n}) = p$.

3.4 Elliptic-curve Cryptography

ECC was first suggested independently by Miller [21] and Koblitz [22] in the mid 1980s. The most popular uses today are Elliptic Curve DSA (ECDSA) and Elliptic Curve Diffie-Hellman (ECDH). The primary advantage of ECC compared to classic arithmetic on prime groups like they are used by RSA [23] or Digital Signature Algorithm (DSA) [24] is the hardness of the mathematical problem behind it.

There exist algorithms for solving the RSA-problem or the discrete logarithm problem (DLP) in finite fields that have subexponential-time complexity [19, Chapter 2]. The index-calculus algorithm for solving DLP in $\mathbb{F}_q$ has a time complexity given in L-notation [19, p. 60] of $L_q[\frac{1}{2}, c]$ with constant $c > 0$ [19, p. 112]. The L-notation is defined as $L_q[\alpha, c] = O\left(e^{(c+o(1))(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha}}\right)$ with the positive constant c and $0 < \alpha < 1$ [19, p. 60, Eqn. 2.3].

However, until now the best known algorithm to solve the elliptic curve discrete logarithm problem (ECDLP) has exponential-time complexity. For general elliptic curves the best known algorithm to solve ECDLP is Pollard's rho algorithm for logarithms [25]. This algorithm has polynominal-time complexity of $O(\sqrt{p})$ in the size of the elliptic curve group. However, the input of complexity descriptions of algorithms is commonly defined in the number of bits. With p being a number of n bits, testing all values of p would take a time of $O(2^n)$. Thus the runtime complexity of Pollard's rho algorithm for general elliptic curves with $p = O(2^n)$ becomes $O(\sqrt{2^n}) = O(2^{n^{\frac{1}{2}}}) = O(2^{\frac{n}{2}})$. This clearly describes an exponential-time complexity algorithm.

Definition 3.7. The *discrete logarithm problem* is defined as finding element x for a given a, g and p, p being a large prime, in the formula:

$$a \equiv g^x \mod p \tag{5}$$

The corresponding problem in elliptic curve groups is called ECDLP.

Definition 3.8. The *elliptic curve discrete logarithm problem* is defined as finding $n \in \mathbb{Z}$ given $P, Q \in E(\mathbb{F}_q)$ in the following function. E is describing the elliptic curve function and $\mathbb{F}_q$ the finite field used for the coordinates.

$$Q = nP \tag{6}$$

Symmetric	Asymmetric (RSA / DLOG)	Asymmetric (Elliptic Curve)
64	816	128
80	1248	160
112	2432	224
128	3248	256
160	5312	320

Table 4: ECRYPT II comparison of key sizes (in bits) at the same security level between symmetric, asymmetric (RSA) and elliptic curve [26].

Due to the hardness of the ECDLP, one can use smaller groups in ECC-based schemes with the same equivalent symmetric security level as compared to schemes based on RSA or DLP. This leads to smaller key sizes and smaller signatures, which is especially beneficial in low-power computing environments. A comparison of key sizes of equal security in symmetric, traditional asymmetric and the elliptic curve setting can be found in Table 4 and takes latest hardware advances and the best algorithms for the cryptographic problems into account.

3.4.1 Traditional Weierstrass Curves

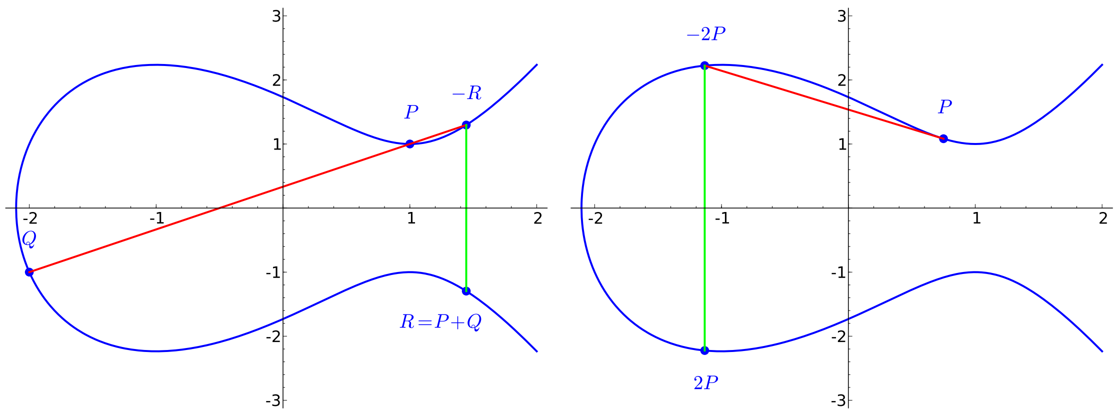
Elliptic curves as mathematical groups, are commonly defined in short Weierstrass form as

$$E(\mathbb{F}_q) = \left\{(x, y) \in {\mathbb{F}_q}^2 : y^2 = x^3 + ax + b\right\} \bigcup \mathcal{O} \quad , \tag{7}$$

with point $\mathcal{O}$ representing the additive identity and $\mathbb{F}_q$ being the field used for the x, y-coordinates. Additionally, to avoid certain attacks the elliptic curves are required to be non-singular. This requires a discriminat of $\Delta = 4a^3 + 27b^2 \neq 0$.

For elliptic curves defined over the real numbers ($\mathbb{R}$), $\mathcal{O}$ can also be seen as a point infinitely far off the x-axis. A visualization of an elliptic curve defined over $\mathbb{R}$ can be seen in Figure 1. However, in ECC a finite field ($\mathbb{F}_q$) is commonly used. $\mathbb{F}_q$ is of prime ($q = p$) or prime power ($q = p^k$) order, where p is prime and k is a positive integer.

A group operation needs to be defined to form a group of a set of elliptic curve points. For elliptic curves this is the addition of two elliptic curve points using the well known *chrod-and-tangent* method. Addition works by mirroring the third intersection of the chord over two points (Q, R) with the curve over the x-axis and can be seen in Figure 1(a). Point doubling, the addition of a point with itself, is defined in a similar fashion, just that the ray already intersects the curve in the point P twice, with the ray being the tangent at that point, as seen in Figure 1(b).



(a) Addition: $P + Q$ (b) Doubling: $2P$

Figure 1: Geometric description of elliptic curve group operations in $E(\mathbb{R})$ for the curve $y^2 = x^3 - 3x + 3$ rendered by Sage[27]. Chord(a) between P and Q, and tangent(b) at point P in red.

While elliptic curves over $\mathbb{R}$ work well for geometric description of the addition and doubling law on elliptic curves, they are unsuitable for cryptographic implementation. Elements of $\mathbb{R}$ are hard to represent and precise computations are expensive for computers. Cryptographic applications use elliptic curves over finite fields. For example, the same elliptic curve as in Figure 1 however over $\mathbb{F}_{43}$, can be seen in Figure 2.

There are various easily computable methods for calculating point addition and point

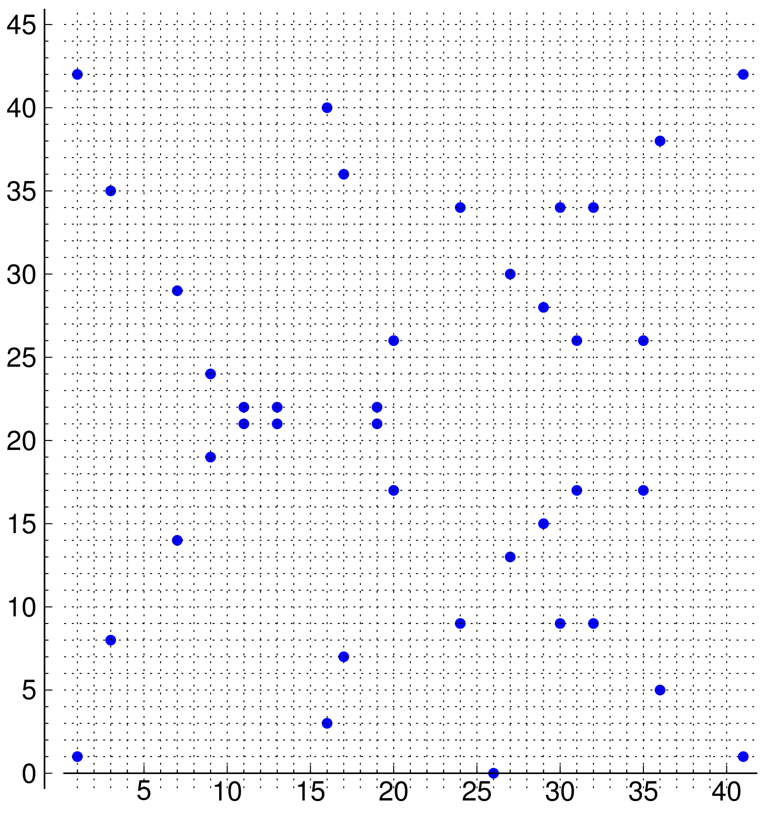


Figure 2: Elliptic curve $E(\mathbb{F}_{43}) : y^2 = x^3 - 3x + 3$ rendered by Sage[27].

doubling for $E(K)$. The points over an elliptic curve together with the elliptic curve operation, the addition of two points, form an Abelian group, $(E(K), +)$. The point at infinity, $\mathcal{O}$, serves as identity element of the group.

The order of the elliptic curve group, $\#E(K)$, is defined as the number of points satisfying the elliptic curve equation E over the field K plus the point at infinity $\mathcal{O}$. For prime fields, the order of an elliptic curve $\#E(\mathbb{F}_p)$ is equal to $q + 1 - t$ with $|t| \leq 2\sqrt{q}$ [28].

Definition 3.9. The order of point P is defined as the smallest k, the number of self-additions required to reach $\mathcal{O}$: $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}} = \mathcal{O}$.

The cyclic group of elliptic curve points is defined by the *generator* or *base point* P. Repeated application of the group operation to P generates all elements of the group. For an elliptic curve group of order n, there exists a point P, which generates the elliptic curve group $E(\mathbb{F}_p) = \{kP : 1 \leqslant k \leqslant n\}$.

Since the discovery of ECC, there have been a wide variety of choices in both the kind of elliptic curves to use and in the underlying field.

The choice of curve parameters and field type is not always apparent and universal. The choice of field—prime field ($\mathbb{F}_p$) or binary extension field ($\mathbb{F}_{2^m}$)—can have a crucial influence on the performance depending on hardware architecture [29]. Bilinear pairings transfer supersingular curves over $\mathbb{F}_q$ to $\mathbb{F}^*_{q^k}$, where the DLP is easier to solve compared to the hard ECDLP. This fact is used by the MOV attack [30]. Some elliptic curves over $\mathbb{F}_{q^n}$ transfer to hyperelliptic curves, where the ECDLP is easier to solve [31]. Elliptic curves over $\mathbb{F}_p$ or $\mathbb{F}_{2^p}$ where p is a prime are not vulnerable to this attack.

In 2008 Koblitz, Koblitz, and Menezes [32] provided an overview on the evolution of ECC. This includes three sections on the selection on secure elliptic curves and many sometimes conflicting properties which are relevant for the security of cryptography systems based on elliptic curves.

Conservative choices for elliptic curves have been standardized by NIST [24] and Brainpool [33] for different security levels. The standards commonly follow the approach of generating random curves in the spectrum of curves not vulnerable to currently known attacks.

Example: ECDSA Signature A popular and widespread used signature algorithm based on ECC is ECDSA [28]. The following description of ECDSA assumes already defined domain parameters: elliptic curve E, finite field used for point coordinates on the curve $\mathbb{F}_q$, the order of the elliptic curve group n and its based point G. The order of the elliptic curve group is described with n.

1. Key Generation
   a) Choose a random private key $x \in \mathbb{Z}_n$.
   b) Compute the corresponding public key $Q = xG$.
2. Signature Generation
   a) Choose a random integer $k \in \mathbb{Z}_n$.
   b) Set $(x_1, y_1) = kG$.
   c) Compute $r = x_1 \mod n$ and repeat from a) if $r = 0$.
   d) Compute hash e of message m with $e = H(m)$.
   e) Set $s = k^{-1}(e + xr) \mod n$ and repeat from a) if $s = 0$.
   f) The final signature of message m is (r, s).
3. Signature Verification

   Verifying signature (r, s) of message m by an entity with public key Q works as follows:
   a) Verify that $r, s \in \mathbb{Z}_n$.
   b) Compute $e = H(m)$.
   c) Set $w = s^{-1} \mod n$.

d) Compute $u_q = ew \mod n$ and $u_2 = rw \mod n$.
e) Set $X = u_1 G + u_2 Q$.
f) Reject the signature if $X = \mathcal{O}$.
g) If $X \neq \mathcal{O}$, convert x_1 of $X = (x_1, y_1)$ to an integer and accept signature if $x_1 = r \mod n$.

3.4.2 Modern Edwards Curves

There are various different ways to define elliptic curves over a field $\mathbb{F}_p$ beside the classic short Weierstrass equation shown earlier. Even when using the short Weierstrass equation there are various forms to calculate the curve arithmetic. Curve arithmetic covers the point representation used in the system. Points can be represented in affine, standard projective, Jacobian projective and more different forms. For each form there are optimized addition formulas. They describe what low level arithmetic operations in the underlying field need to be carried out to compute one group operation, point addition or point doubling. The different addition formulas often differ in their computational complexity. There has been steady development to find representations and formulas which minimse the number of operations in the underlying field, or avoid expensive field operations like inversions. The different choices of doing arithmetic on elliptic curves are explained in great detail in [20, Chapter 13].

The most recent development in this area is the use of Edwards curves [34] for cryptographic applications. Bernstein and Lange [35] define modern Edwards curves as $x^2 + y^2 = c^2 \left(1 + dx^2y^2\right)$, with $cd \left(1 - dc^4\right) \neq 0$. The example curve—Curve1174—is shown in Figure 3. In contrast to Weierstrass curves, the identity element of a group of points on the curve is not a special point which needs to be handled as edge case, but instead is a point on the curve, i.e. $\mathcal{O} = (0, c)$.

The advantage of Edwards curves and its associated arithmetic is not only in processing speed but also in its security. Edwards curves provide the fastest group operation among the currently known curve forms. Their security advantage lies in the completeness of the addition formulas. This means they work with any pair of input points, including the identity point or two identical points. In contrast to short Weierstrass curves and their associated addition formulas, where there are different formulas for the different cases of doubling and addition involving the identity element $\mathcal{O}$. This opens a timing side-channel in the computations on the elliptic curve which increases the attack surface.

A detailed comparison of the costs of group operations in the different representations is given in [35, Section 4]. In addition, Weierstrass curves and Edwards curves can be naturally transformed into each other. Transformation may require a change of the underlying field [35].

3.5 Pairing-based Cryptography

Initially pairing-based cryptography (PBC) was used for cryptanalytic purposes only. The most prominent example for this is the MOV attack [30] which utilizes bilinear pairings to

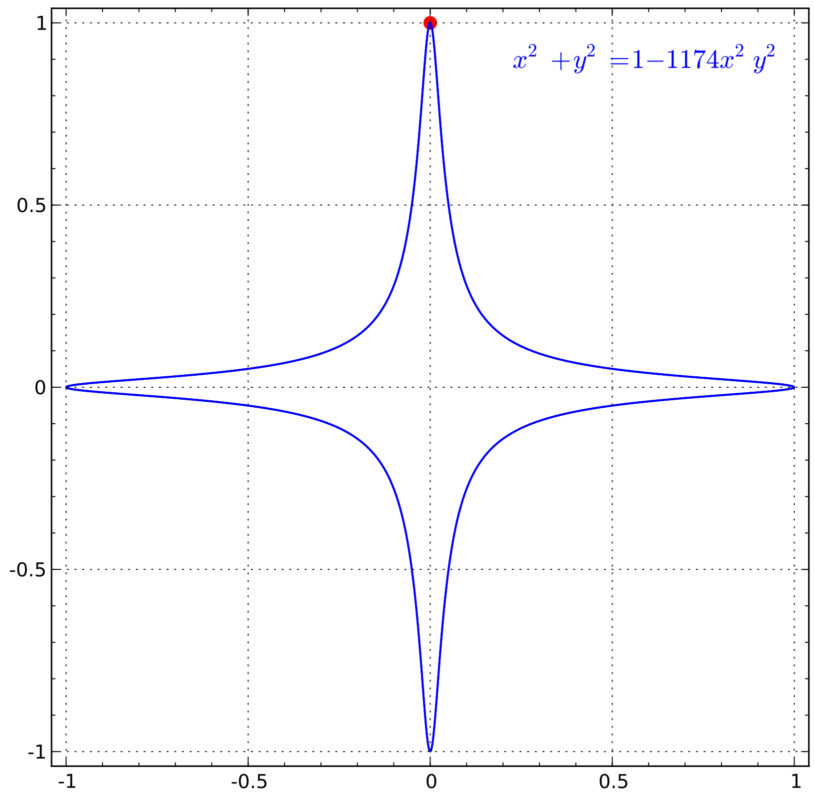


Figure 3: The Edwards curve Curve1174 over $\mathbb{R}$. The identity element of the elliptic curve group is the red highlighted point $\mathcal{O} = (0, 1)$.

reduce the discrete logarithm problem in the elliptic curve group $E(F_p)$ of a supersingular curve to a classic discrete logarithm problem in an extension field $\mathbb{F}_{p^k}$.

The DLP in $\mathbb{F}_{p^k}$ is easier to solve than the ECDLP. This means the security of the ECC system is no longer defined by the size of the elliptic curve group of $E(F_p)$ but of the size of the extension field $\mathbb{F}_{p^k}$. Care needs to be taken to choose p big enough that the system remains secure with respect to the size of the extension field and the classic DLP.

Joux first suggested to use bilinear pairings for cryptographic purposes in 2000 [36][1]. He showed that bilinear pairings not only allow the breaking of existing ECC system which use certain curves but that they also help to build new cryptographic protocols that have not been possible before. Joux proposed a protocol for computing a shared secret among three parties, also known as tripartite Diffie-Helman, in one round.

Shortly after Boneh and Franklin introduced the first realization of IBE using PBC [9]. The concept of IBE was already proposed by Shamir in 1985 but only the advances in PBC

[1]He later provided a revised version with [37].

allowed for a practical efficient implementation of it.

Definition 3.10. A bilinear pairing is a map $\hat{e}$ from elements of two groups, G_1 and G_2, in a target group G_T: $\hat{e} : G_1 \times G_2 \longrightarrow G_T$. With $a, b \in \mathbb{Z}$, $P \in G_1$ and $Q \in G_2$, it has the following properties:

- Bilinearity:

$$\hat{e}(aP, bQ) = \hat{e}(P, bQ)^a = \hat{e}(aP, Q)^b = \hat{e}(P, Q)^{ab} \tag{8}$$

- Non-degeneracy:

$$\forall P \in G_1 : P \neq 0 \Longrightarrow \exists Q \in G_2 \wedge \hat{e}(P, Q) \neq 1 \tag{9}$$
$$\forall Q \in G_2 : Q \neq 0 \Longrightarrow \exists P \in G_1 \wedge \hat{e}(P, Q) \neq 1 \tag{10}$$

- Efficiently computable: for bilinear pairings to be useful in cryptographic protocols, efficient implementations computing the pairing must exist. This is the case for the Weil- and Tate-pairing using Miller's algorithm [38]. Further discussion on efficiency can be found in [39].

In addition to Definition 3.10, one distinguishes two forms of pairings commonly found in the literature:

- *Symmetric pairings* where two elements of the same group are mapped into a target group: $\hat{e} : G_1 \times G_1 \longrightarrow G_T$
- *Asymmetric pairings* where two elements of different groups are mapped into a target group: $\hat{e} : G_1 \times G_2 \longrightarrow G_T$

The symmetric case can be seen as a special case of the asymmetric one, with $G_1 = G_2$ [40].

The only way to implement *symmetric* pairings is using supersingular curves over $\mathbb{F}_q$. q describes the number of elements in the field and has to be big enough, considering most supersingular curves have a low embedding degree k, typically $k \leq 6$. G_1 and G_2 are groups of points on an elliptic curve and G_T a multiplicative group in a finite field. In case of $G_1 = E(\mathbb{F}_q)$, G_T is the multiplicative group $\mathbb{F}^*_{q^k}$ [40]. Since there are subexponential time algorithms to solve the DLP in a multiplicative finite field group, q needs to be big enough so the system will be resilient against the MOV attack at the desired security level.

A popular implementation of *asymmetric* pairings is the use of BN-curves [41], which are special non-supersingular curves with a relative high embedding degree ($k = 10$). In this case the new group G_2 is the subgroup of the group of points of $E(\mathbb{F}_{q^k})$.

As for the actual pairing function, usually called e or $\hat{e}$, there are two choices: the Weil and Tate pairings. Both are computed via Miller's algorithm, however the Tate pairing can provide performance benefits over the Weil pairing [42].

Example: BLS Signature One of the simplest examples of bilinear pairings used in cryptography is the BLS [43] signature. It consist of the following three functions, when used with symmetric pairings:

1. *Key Generation*: Choose random $x \in \mathbb{Z}_p$, the user's private key and publish $v = g^x$, the corresponding public key. g is the generator of G_1.
2. *Signature Generation*: A signature is generated by first hashing a message m, $h = H(m)$, and computing the signature $\sigma = h^x$.
3. *Signature Verification*: Given a public key v and a signature σ, message m is verified by asserting $\hat{e}(\sigma, g) = \hat{e}(H(m), v)$.

The verification holds because with $\sigma = H(m)^x$ and $v = g^x$, $\hat{e}(\sigma, g) = \hat{e}(H(m)^x, g) = \hat{e}(H(m), g^x) = \hat{e}(H(m), v)$.

This signature does not only generate smaller signatures compared to standard ECDSA, it is also much simpler in its description and does not require random numbers for signature generation.

4 Experiment

To test the viability of identity-based signatures (IBSs) for constrained devices, three different types of IBSs are implemented and tested for performance. In case of elliptic curve based signature schemes, different curves are tested to explore how the choice of the curve influences the performance. For comparison, well established classic public key signatures like RSA and Elliptic Curve DSA (ECDSA) are included in the overall comparison. For the elliptic curve based schemes, in addition, different elliptic curves, beyond publicly standardized curves, are evaluated.

4.1 Schemes

In this section, the evaluated signature schemes are shortly introduced and the ID-based signature schemes are defined, the way they have been implemented later. The computational complexity of signature generation and verification, and the signature size are given in terms of group operations and group elements respectively.

4.1.1 Classic Asymmetric Signature Schemes

For simple comparison against commonly know asymmetric signature schemes, RSA [23] and ECDSA [28] are part of the benchmark. For these schemes the standard implementation in RELIC [5] is used.

4.1.2 SH-IBS (RSA based scheme)

This is the first proposal for an IBS scheme by Shamir from 1985 [3]. It is based on the RSA crypto system and its security depends on the hardness of integer factorization in the RSA problem.

Setup: The setup is similar to standard RSA crypto system setup. For the setup given security parameter k proceed with the following steps:

1. Generate two distinct primes p and q at random with $2^{\frac{k-1}{2}} < p, q < 2^{\frac{k}{2}}$
2. Calculate n and generate the master public key, e, via:

$$n = p \cdot q \tag{11}$$
$$e \in [1 : \varphi(n)] \wedge e \perp \varphi(n) \tag{12}$$

   e is relative prime to $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$

3. Calculate the master private key, d:

$$d = e^{-1} \mod \varphi(n) \tag{13}$$

4. Choose two commonly known secure hash functions: H_1 maps opaque user identification strings to values in $\mathbb{Z}_n$ and H_2 is used as part of signature generation and verification:

$$H_1 : \{0,1\}^* \to \mathbb{Z}_n \tag{14}$$
$$H_2 : \mathbb{Z}_n \times \{0,1\}^* \to \mathbb{Z}_n \tag{15}$$

5. Publish public system parameters:

$$\langle n, e, H_1, H_2 \rangle \tag{16}$$

Key Extraction: To generate the private key, s_{ID} for a user with the identity ID proceed with the following step:

1. Calculate user private key with:

$$s_{ID} = H_1(ID)^d \mod n \tag{17}$$

Signature Generation: For generating the signature for message $m \in \{0,1\}^*$ do:

1. Generate random $r \in \mathbb{Z}_n$.

2. Calculate the signature $\sigma = (s, t)$:

$$t = r^e \quad \mod n \tag{18}$$
$$s = s_{ID} \cdot r^{H_2(t,m)} \quad \mod n \tag{19}$$

Signature Verification: To verify signature $\sigma = (s, t)$ for message m and user identity ID belonging to that message, do the following:

1. Check whether the equation holds:

$$s^e \overset{?}{=} H_1(ID) \cdot t^{H_2(t,m)} \quad \mod n \tag{20}$$

Complexity Overview: As can be seen in the last two paragraphs, signature generation requires two modular exponentiations in $\mathbb{Z}_N$ and signature verification one modular multi-exponentiation in $\mathbb{Z}_N$. The cost of hashing and multiplication is negligible compared to the exponentiations. The signature defined as $\sigma = (s, t)$ requires two elements of $\mathbb{Z}_N$.

4.1.3 vBNN-IBS (Elliptic-curve based scheme)

vBNN-IBS is a ID-based signature scheme described by Cao, Kou, Dang, *et al.* as part of IMBAS [44].

Setup: To initialize the system for security parameter k, take the following steps:

1. Chose an elliptic curve with the parameters $E/\mathbb{F}_q$, P and p satisfying the security parameter k.
2. Generate random master secret key $x \in \mathbb{Z}_p$ and set the master public key, $P_0 = xP$.
3. Define two cryptographic hash functions, H_1 and H_2:

$$H_1 : \{0,1\}^* \times \mathbb{G} \rightarrow \mathbb{Z}_p \tag{21}$$
$$H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{Z}_p \tag{22}$$

4. Publish public system parameters:

$$\langle E/\mathbb{F}_q, P, p, P_0, H_1, H_2 \rangle \tag{23}$$

Key Extraction: To generate the private key, s_{ID}, for a user with the identity ID, carry on with the following steps:

1. Calculate a random $r \in \mathbb{Z}_p$ and compute $R = rP$.

2. Using the master secret key x, calculate:

$$c = H_1(ID, R) \tag{24}$$
$$s = r + cx \tag{25}$$

3. The private key for the user with the identity ID is $s_{ID} = (R, s)$.

Signature Generation: For generating the signature σ for message $m \in \{0,1\}^*$ do:

1. Generate random $y \in \mathbb{Z}_p$ and compute $Y = yP$.
2. Compute the following:

$$h = H_2(ID, m, R, Y) \tag{26}$$
$$z = y + hs \tag{27}$$

The final signature for message m is $\sigma = (R, h, z)$.

Signature Verification: To verify if message m from a user with the identity ID is correctly authenticated by signature $\sigma = (R, h, z)$, proceed with:

1. Compute the following:

$$c = H_1(ID, R) \tag{28}$$
$$T = zP - h(R + cP_0) \tag{29}$$

2. To verify the signature, check whether the following equation holds:

$$h \stackrel{?}{=} H_2(ID, m, R, T) \tag{30}$$

Complexity Overview: Signature generation comes with the cost of one scalar multiplication in $E(\mathbb{F}_q)$ from step 1 and signature verification costs 3 scalar multiplications in $E(\mathbb{F}_q)$ from equation 26. The signature size, with $\sigma = (R, h, z)$, is one element of $E(\mathbb{F}_q)$ and two elements of $\mathbb{Z}_q$.

4.1.4 TSO-IBS (Pairing based scheme)

Under TSO-IBS [45], we consider the ID-based signature scheme proposed by Tso, Gu, Okamoto, *et al.* Contrary to the other schemes evaluated in this work, TSO-IBS provides message recovery. This means that the signature generation algorithm produces a signature which already includes the message or where the original message can be extracted from by any receiver. This aims to reduce the overall overhead of the signature on the communication traffic.

In addition, TSO-IBS is based on symmetric bilinear pairings. They propose two schemes, one for fixed sized messages and another one for arbitrary sized messages. In this section the first one will be described.

Setup: To initialize the system proceed as follows:

1. G_1 and G_2 are two cyclic groups of order q, $|q| = l_1 + l_2$
2. There is a symmetric bilinear pairing: $\hat{e} : G_1 \times G_1 \longrightarrow G_2$
3. Generate random $s \in \mathbb{Z}_q^*$ as master secret key.
4. Set $P_{pub} = sP$ as the master public key.
5. Calculate $\mu = \hat{e}(P, P)$.
6. Define four cryptographic hash functions H, H_1, F_1 and F_2:

$$H : \{0,1\}^* \longrightarrow \mathbb{Z}_p^* \tag{31}$$
$$H_1 : \{0,1\}^* \longrightarrow \{0,1\}^{l_1+l_2} \tag{32}$$
$$F_1 : \{0,1\}^{l_1} \longrightarrow \{0,1\}^{l_2} \tag{33}$$
$$F_2 : \{0,1\}^{l_2} \longrightarrow \{0,1\}^{l_1} \tag{34}$$

7. Publish the public system parameters:

$$\langle G_1, G_2, \hat{e}, q, P, P_{pub}, \mu, H, H_1, F_1, F_2, l_1, l_2 \rangle \tag{35}$$

Key Extraction: To generate the private key, s_{ID} for a user with the identity ID proceed with the following step:

1. $s_{ID} = (H(ID) + s)^{-1} P$

Signature Generation: For signing message $m \in \{0,1\}^{l_1}$ do:

1. Generate random $r_1 \in \mathbb{Z}_q^*$.
2. Compute $\alpha = H_1(ID, \mu^{r_1}) \in \{0,1\}^{l_1+l_2}$.
3. Compute $\beta = F_1(m) || (F_2(F_1(m)) \bigoplus m)$ and $r_2 = [\alpha \bigoplus \beta]$.
4. Compute $U = (r_1 + r_2) s_{ID}$.

The final signature for message m is $\sigma = (r_2, U)$.

Signature Verification: To verify a signature $\sigma = (r_2, U)$ and recover the message $\tilde{m}$ from a user with identity ID, a verifier does the following steps. $\tilde{\alpha}$, $\tilde{\beta}$ and $\tilde{m}$ are the recovered variables α, β and m from signature generation respectively.

1. Calculate public key of user via $P_{ID} = H(ID)P + P_{Pub}$.
2. Compute $\tilde{\alpha} = H_1(ID, \hat{e}(U, P_{ID}) \cdot \mu^{-r_2})$.
3. Compute $\tilde{\beta} = r_2 \bigoplus \tilde{\alpha}$.
4. Recover message $\tilde{m} = |\tilde{\beta}|_{l_1} \bigoplus F_2({}_{l_2}|\tilde{\beta}|)$.
5. The signature σ and the recovered message $\tilde{m}$ are valid, if ${}_{l_2}|\tilde{\beta}| = F_1(\tilde{m})$.

Correctness of Signature: Essential for the correctness of the signature is the recovery of $\tilde{\alpha}$ from public information and its equality to α. For this we show that $\hat{e}(U, P_{ID} \cdot \mu^{-r_2}) = \mu^{r_1}$.

$$\begin{aligned}
&\hat{e}\left(U, P_{ID}\right) \cdot \mu^{-r_2} && (36)\\
=&\hat{e}\left((r_1 + r_2) \cdot s_{ID}, P_{ID}\right) \cdot \mu^{-r_2} && (37)\\
=&\hat{e}\left(s_{ID}, P_{ID}\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (38)\\
=&\hat{e}\left((H(ID) + s)^{-1} \cdot P, P_{ID}\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (39)\\
=&\hat{e}\left((H(ID) + s)^{-1} \cdot P, H(ID) \cdot P + P_{Pub}\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (40)\\
=&\hat{e}\left((H(ID) + s)^{-1} \cdot P, H(ID) \cdot P + s \cdot P\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (41)\\
=&\hat{e}\left((H(ID) + s)^{-1} \cdot P, (H(ID) + s) \cdot P\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (42)\\
=&\hat{e}\left(P, P\right)^{(r_1+r_2)} \cdot \mu^{-r_2} && (43)\\
=&\mu^{(r_1+r_2)} \cdot \mu^{-r_2} && (44)\\
=&\mu^{r_1} && (45)
\end{aligned}$$

Complexity Overview: The generation of signatures costs one exponentiation in G_2, from the μ^{r_1} in step 1, and one scalar multiplication in $E(\mathbb{F}_q)$ from step 4. Signature verification is more expensive, with one scalar multiplication in $E(\mathbb{F}_q)$ in step 1, one pairing computation and one exponentiation in G_2 in step 2. The signature size, including the original message, is one lement in G_1 plus $|q|$, the bit length of the binary representation of q.

4.2 Signature Size and Performance

In this section, an overview about the asymmetric signature schemes at hand is given with regard to the abstract computational complexity for signature generation and signature verification in Table 5. The complexity is expressed in abstract operations on the mathematical objects. Furthermore, the size of produced signatures is quantified by the number of elements from the groups used in the signature. This table is similar to table 1 in [6] from Kiltz and Neven, although with different signature schemes.

The operations used in the comparison are exponentiations (*exp.*) in $\mathbb{Z}_N$, $\mathbb{Z}_q$ or $E(\mathbb{F}_q)$, modular multiplicative inverse (*mod. inv.*) in $\mathbb{Z}_q$, modular multi-exponentiation (*mexp.*) in $\mathbb{Z}_N$ and pairing computation ($\hat{e}(\cdot,\cdot)$). Computing the modular multiplicative inverse is the most computationally expensive operation. Modular multi-exponentiation is an optimization for cases where multiple elements are exponentiated with the same power. Pairing computations are more expensive than exponentiations in the elliptic curve group $E(\mathbb{F}_q)$.

Group elements for signature size are $\mathbb{Z}_q$ or $\mathbb{Z}_N$ for numbers in a prime group or a group of integers modulo some large N where N is the product of two large primes or elements of $E(\mathbb{F}_q)$, which can be compressed to nearly one element in $\mathbb{F}_q$.

Scheme	Complexity		Signature size
	Signing	Verification	
RSA [23]	1 exp. in $\mathbb{Z}_N$	1 exp. in $\mathbb{Z}_N$	$\mathbb{Z}_N$
ECDSA [28]	1 exp. in $E(\mathbb{F}_q)$ 1 mod. inv. in $\mathbb{Z}_q$	2 exp. in $E(\mathbb{F}_q)$ 1 mod. inv. in $\mathbb{Z}_q$	$\mathbb{Z}_q \times \mathbb{Z}_q$
SH–IBS [3]	2 exp. in $\mathbb{Z}_N$	1 mexp. in $\mathbb{Z}_N$	$\mathbb{Z}_N \times \mathbb{Z}_N$
vBNN-IBS [44]	1 exp. in $E(\mathbb{F}_q)$	3 exp. in $E(\mathbb{F}_q)$	$E(\mathbb{F}_q) \times \mathbb{Z}_q \times \mathbb{Z}_q$
TSO-IBS [45]	1 exp. in $G_T = \mathbb{F}^*_{q^k}$ 1 exp. in $G_1 = E(\mathbb{F}_q)$	1 exp. in $G_T = \mathbb{F}^*_{q^k}$ 1 exp. in $G_1 = E(\mathbb{F}_q)$ 1 pairing $\hat{e}(\cdot,\cdot)$	$G_1 \times \lvert q \rvert$

Note: Multiplications in the additive group $E(\mathbb{F}_p)$ are referred to as exponentiations.

Table 5: Comparison of computational complexity and signature size of classic asymmetric signatures and ID-based signatures.

Table 5 only gives an abstract comparison of the time and space complexity for the different signature schemes. The actual complexity depends on the size of N, the size and the kind of field in use for the elliptic curves, $\mathbb{F}_q$, and the actual pairing that is used. In addition, the required sizes for RSA based schemes and elliptic curve based schemes scale differently with the respective symmetric security level (see Table 4).

4.3 Evaluation Architecture

4.3.1 Benchmark Foundation

As a first step the schemes are implemented in Charm [4], a Python framework for cryptographic prototyping.

Charm enables nearly direct transformation from the mathematical notation used in the papers to Python source code. However, it is to note that Charm uses multiplicative notation for elliptic curve group operations. This means that elliptic curve point addition is carried out, rather counter-intuitive, with the *-operator and multiplication with the **-operator.

```python
from charm.toolbox.ecgroup import ECGroup, ZR, G
from charm.toolbox.PKSig import PKSig

class vBNN_IBS(PKSig):
  def __init__(self, groupObj):
    global group,H1,H2
    group = groupObj
    H1 = lambda ID, R: group.hash((ID, R))
    H2 = lambda ID, m, R, Y: group.hash((ID, m, R, Y))

  def setup(self):
    x, P = group.random(), group.random(G)
    P_0 = (P ** x)
    ssk = x
    spk = {'P': P, 'P_0': P_0}
    return (spk, ssk)

  def keygen(self, spk, ssk, ID):
    x = ssk
    r = group.random()
    R = spk['P'] ** r
    c = H1(ID, R)
    s = r + (c * x)
    return {'R': R, 's': s}

  def sign(self, spk, ID_A, m, Pri_A):
    y = group.random()
    Y = spk['P'] ** y
    h = H2(ID_A, m, Pri_A['R'], (Y))
    z = y + (h * Pri_A['s'])
    return (Pri_A['R'], h, z)

  def verify(self, spk, ID_A, m, sig):
    (R, h, z) = sig
    c = H1(ID_A, R)
    Y = (spk['P'] ** z) / ((R * (spk['P_0'] ** c)) ** h)
    if h == H2(ID_A, m, R, Y):
      return True
    return False
```

Listing 1: Implementation of vBNN-IBS using Charm.

Implementing the schemes in Charm provides deeper understanding of the algorithms and test data to validate the target implementation in a more efficient environment, with better suitability for constrained devices. An example showing the straightforward implementation of cryptographic schemes can be seen in Listing 1. This shows the implementation of vBNN-IBS [44] in Python using the Charm library.

The code used for the following benchmark however is written in C/C++ using the RELIC toolkit [5]. The RELIC toolkit is a C library, which targets low-resource devices like wireless sensor nodes and runs on a wide range of architectures, namely AVR, MSP, ARM, x86 and x86_64.

4.3.2 Test Candidates

The signature schemes presented in Subsection 4.1 are used as candidates for the following benchmark. The benchmark measures the runtime performance of signature verification and signature generation, and the size of the signature of the ID-based signature schemes. The RSA-based scheme, SH–IBS, is compared to classic RSA for different security parameters.

For the elliptic-curve based schemes the situation is more diverse. Here the ID-based signature schemes are not only compared to classic ECDSA, but also with different elliptic curves. This includes standard curves from NSA Suite B [46] like NIST P-256, and also more modern curves like Curve1174 [47], Curve25519 [48] and Curve383187 [49]. These also cover different security levels. The full list of curves used in the benchmark and their security properties are listed in Table 6.

Name	$\|\mathbb{F}_q\|$ (bits)	Symmetric Security Level (bits)
Curve1174	251	126
Curve25519	256	128
NIST-P256	256	128
Curve383187	384	192
NIST-P384	384	192
SS-P1536	1536	128

Table 6: Elliptic curves used in the benchmark and their respective symmetric security level.

4.3.3 Implementation

The RELIC toolkit only provides a C level API to the programmer. The initial implementation in Charm has shown, that a high-level API allows a nearly direct transfer of cryptographic schemes from papers into code. To ease development of cryptographic schemes with the RELIC toolkit, we created C++ wrapper classes for the basic RELIC types like `bn_t`(arbitrary precision numbers), `ec_t`(elliptic curve points) and `g1_t`, `g2_t`, `gt_t`(group elements for pairing-based protocols). These wrapper classes deal with the otherwise manual and error-prone memory management and overload mathematical operators which allow cryptographic schemes from papers to be implemented in code in a straightforward fashion.

Example code for vBNN-IBS implemented using the C++ wrapper for RELIC is shown in Listing 2.

```cpp
using namespace relic;
assert(ep_param_set_any() == STS_OK);

// === Setup ===
bn n, x;
ec P, P_0;
ec_curve_get_ord(n);
x = bn::random();
P = ec::random();
P_0 = P * x;

// === Key Extraction ===
std::string user = ”alice@wonderland.lit”;
bn r = bn::random(), s;
ec R = P * r;
s = (r + hash_mod_bn(n, user, R) * x) % n;

// === Signature Generation ===
std::string message = ”Art␣thou␣not␣Romeo,␣and␣a␣Montague?”;
bn y = bn::random(), h, z;
ec Y;
Y = P * y;
h = hash_mod_bn(n, user, message, R, Y);
z = (y + h * s) % n;

// === Signature Verification ===
bn c = hash_mod_bn(n, user, R);
ec Z = (P * z) - (R + P_0 * c) * h;
assert((h == hash_mod_bn(n, user, message, R, Z)) && ”vBNN-IBS␣verification␣failed.”);
```

Listing 2: Implementation of vBNN-IBS using the RELIC toolkit C++ wrapper.

5 Results

We use the clang compiler (version 3.5.0) to build the benchmark and run it in two different software and hardware environments, shown in Table 7. We used version `r1861` of the RELIC toolkit in this benchmark. We run each benchmark program five times per test combinations and measure the wall-clock time in each run. In addition, the CPU cycle count is measured for signature generation and verification of each test candidate. Within the benchmark program, the RELIC toolkit is initialised and keys are generated. Afterwards we measure the time for generating 100 signatures and measure the time for verifying 100 signatures. It is worth mentioning that the measurements not only cover the mathematical algorithms, but also the random number generation used during signature generation.

Cycle counts are measured using the Time Stamp Counter (TSC) on the Intel and ARM platforms. The TSC is a hardware counter available through a CPU register which increments at CPU clock frequency. It is important to note that the TSC is not synchronized across all CPU cores on multi-core CPUs and its clock speed may change due to power saving features of modern CPUs. The TSC is read-only and cannot be saved across events like context switching which results in measurement variances.

Both, the time and the cycle count measured, are subject to some variations due to the fact that the benchmark is carried out on multitasking operating systems. However, we

	Desktop	Embedded
Device	Laptop	Raspberry Pi
OS	Mac OS X 10.9.3	Debian 3.10.11-1+rpi7
Architecture	Intel	ARM
CPU	Corei7	ARM1176
Word size	64 bit	32 bit
Clock speed	2 GHz	800 MHz
L1 Cache	64 kB	32 kB
L2 Cache	256 kB	(256 kB)
L3 Cache	6 MB	—

Note: The L2 Cache is used by the GPU on the Raspberry Pi and therefore is not available to the CPU.

Table 7: Test environments used for the benchmark.

Asymmetric Security (Bits)	RSA (bits)	SH-IBS (bits)
768	768	1536
1024	1024	2048
1536	1536	3072
2048	2048	4096

Table 8: Signature size measurements for RSA and SH–IBS.

ran the benchmarks on the test systems with minimal additional system load to reduce these variations.

The best results of all five runs for each signature scheme are taken in comparison. We run each test five times to have stable best results since tests have shown executing them ten or more times did not improve the best results. The best result has the smallest time and cycle measurements. They suggest as little as possible noise activity on the system. Next, we present the results of the benchmark for both the RSA-based schemes and the elliptic curve based schemes. We finish with a summarizing comparison of all ID-based signature schemes with regard to their signature size and execution performance.

The practical measurements for signature sizes in RSA-based schemes as shown in Table 8 reflect the abstract signature sizes from Table 5, with SH–IBS needing twice as much space for the signatures compared to classic RSA. The computational performance for signature verification of SH–IBS is worse than for RSA because the verification for SH–IBS signatures uses an exponentiation by an uncontrollable hash value. However, RSA can select a relatively small e thereby move a performance advantage to the signature verification

Asym. Security	RSA		SH-IBS	
$\lvert\mathbb{Z}_N\rvert = 2^n$	Signing (μs)	Verification (μs)	Signing (μs)	Verification (μs)
768	37,791	1024	9993	10,647
1024	80,252	1601	15,683	17,070
1536	242,409	3227	31,488	34,686
2048	545,647	5358	53,405	59,945

Table 9: Timings on embedded platform (32 Bit, ARM) for signature verification and signature generation of RSA and SH–IBS.

Asym. Security	RSA		SH-IBS	
$\lvert\mathbb{Z}_N\rvert = 2^n$	Signing (μs)	Verification (μs)	Signing (μs)	Verification (μs)
768	2294	64	541	614
1024	3898	81	774	873
1536	7842	115	1045	1161
2048	15,663	167	1538	1716

Table 10: Timings on desktop platform (64 Bit, Core i7) for signature verification and signature generation of RSA and SH–IBS.

process in a classic RSA cryptosystem. This does not work for SH–IBS. The benchmark timings for RSA and SH–IBS are shown in Table 9 for the embedded platform and Table 10 for the desktop platform.

Comparing the performance between the embedded ARM and the desktop Intel platform with increasing security level it shows the following observations.

The performance scales worse on the embedded ARM platform, with the ratio for verification between the highest and lowest security level being 5.4 for embedded and 2.8 for desktop. This is likely due to the lower word size of the ARM (32 bit) which handicaps cryptographic schemes requiring big numbers for security. Similar behavior for RSA has been shown by Gura, Patel, Wander, *et al.* [8] on low performance embedded devices.

An interesting observation is that SH-IBS signature generation and verification show a better performance than the signature generation of RSA. The reason for this has not yet been analyzed in all detail.

The benchmark results for the elliptic curve based signature schemes are shown in Table 11 for the signature size and in Table 12 and Table 13 for the signature processing performance on the embedded and desktop platforms respectively. Values for SS-P1536 are in set in parentheses when used with ECDAS and vBNN-IBS because due to the MOV attack supersingular curves are generally avoided in ECC schemes that do not require them.

Elliptic Curve	ECDSA (Bytes)	vBNN-IBS (Bytes)	TSO-IBS (Bytes)
Curve1174	63	85	N/A
Curve25519	64	85	N/A
NIST-P256	64	85	N/A
Curve383187	96	117	N/A
NIST-P384	96	117	N/A
SS-P1536	(64)	(245)	225

Table 11: Signature sizes for ECDSA, vBNN-IBS and TSO-IBS.

Compared to the classic signature ECDSA, vBNN-IBS shows ~ 1.6 times worse performance for signature generation and verification. The pairing based TSO-IBS shows the worst performance of all, mainly due to its use of symmetric pairing and of large super singular curves.

To conclude this section, an overall comparison of all signature schemes is given. Figure 4 shows the signature size for the ID-based signatures of the different identity-based signature (IBS) schemes at tested security levels. The asymmetric security levels has been roughly converted to their respective symmetric security according to Table 4 from ECRYPT II. Here, well known RSA and elliptic curve cryptography (ECC) signature size scalability can be seen with RSA-based SH–IBS showing exponential scalability with increasing

Elliptic Curve	ECDSA		vBNN-IBS		TSO-IBS	
	Sig. (μs)	Ver. (μs)	Sig. (μs)	Ver. (μs)	Sig. (μs)	Ver. (μs)
Curve1174	13,003	33,842	29,549	57,168	N/A	N/A
Curve25519	13,920	37,574	30,067	57,034	N/A	N/A
NIST-P256	12,632	32,439	26,026	48,618	N/A	N/A
Curve383187	29,800	80,536	71,972	120,170	N/A	N/A
NIST-P384	27,157	69,805	61,243	100,696	N/A	N/A
SS-P1536	162,237	464,844	423,200	859,995	480,268	1,136,839

Table 12: Timings on embedded platform for verification and signing of ECDSA, vBNN-IBS and TSO-IBS.

Elliptic Curve	ECDSA		vBNN-IBS		TSO-IBS	
	Sig. (μs)	Ver. (μs)	Sig. (μs)	Ver. (μs)	Sig. (μs)	Ver. (μs)
Curve1174	1680	4127	3478	6186	N/A	N/A
Curve25519	1789	4294	3575	6542	N/A	N/A
NIST-P256	1601	3870	3424	5604	N/A	N/A
Curve383187	2751	6728	6181	8894	N/A	N/A
NIST-P384	2327	5626	5587	7676	N/A	N/A
SS-P1536	5092	14,506	20,363	26,606	14,356	32,932

Table 13: Timings on desktop platform for verification and signing of ECDSA, vBNN-IBS and TSO-IBS.

symmetric security level and ECC-based vBNN-IBS with linear scalability.

Figure 5 and Figure 7 show the performance results on the embedded platform while Figure 6 and Figure 8 show the results for the desktop platform.

For the signature generation and verification performance of SH–IBS scales worse than vBNN-IBS. SH–IBS only shows a performance advantage at rather low security levels to around 80 bits symmetric security. For the pairing based TSO-IBS, little can be said about its scalability, since it was only tested in one configuration. This is due to the RELIC toolkit, which has only one elliptic curve available suitable for symmetric pairings, being the supersingular curve SS-P1536. However, the single configuration showed the worst performance.

Comparing the timings for the 32 bit ARM embedded platform Figure 5 and the timings for the 64 bit Intel desktop platform Figure 6, ECC based vBNN-IBS shows a better performance compared to RSA based SH-IBS schemes on the embedded platform from ~90 bit security level upwards. The point of equal performance between SH-IBS and vBNN-IBS on the desktop platform is higher as Figure 6 shows. The point of equal performance cannot be seen because we stopped bechmarking SH-IBS at 2048 bit asymmetric security.

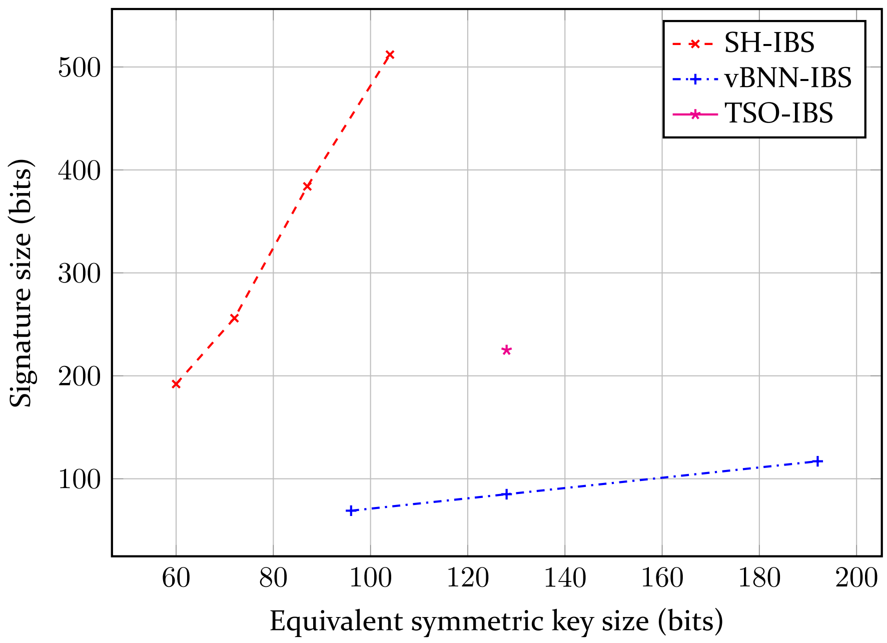


Figure 4: Signature size comparison of the ID-based signatures from the benchmark results.

6 Conclusions & Outlook

All in all this project shows elliptic curve cryptography (ECC) based identity-based signature (IBS) as a valid choice for providing authentication in embedded scenarios. ECC provides better scalability in key size and processing performance with rising symmetric security level, even more so on embedded platforms with smaller word size, like the Raspberry Pi. Good performance and scalability on low power and cost embedded platforms is essential for security solutions for the Internet of Things (IoT) to ensure widespread adoption.

Pairing-based cryptography (PBC) allows to construct new cryptographic schemes which have not been possible without it. The rather high level abstract description of bilinear pairings enables researchers to build new schemes without knowing details about the diverse implementations of pairings. However, the performance of these schemes is more complex to determine. Galbraith, Paterson, and Smart [40] discuss this problem in great detail. Targeting higher performance for pairing based schemes sometimes come with more risky choices of elliptic curves which later turn out be less secure than assumed [50]. In addition, most schemes proposed in scientific papers are described using symmetric pairings, which commonly are less performant than schemes using asymmetric pairings.

Basic ECC on the other hand has been studied for a longer time and while still being in active research, there is a constant search among researchers for fast *and* secure parameter choices and implementation of cryptographic schemes. An area of particular interest is

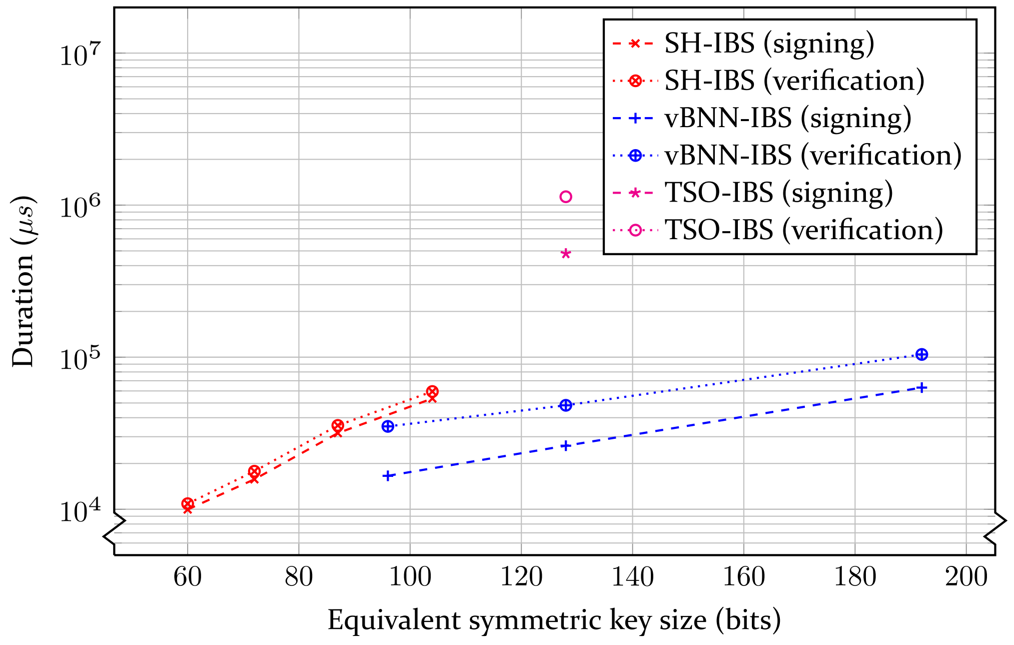


Figure 5: Comparison of signature generation and signature verification timings on the embedded platform.

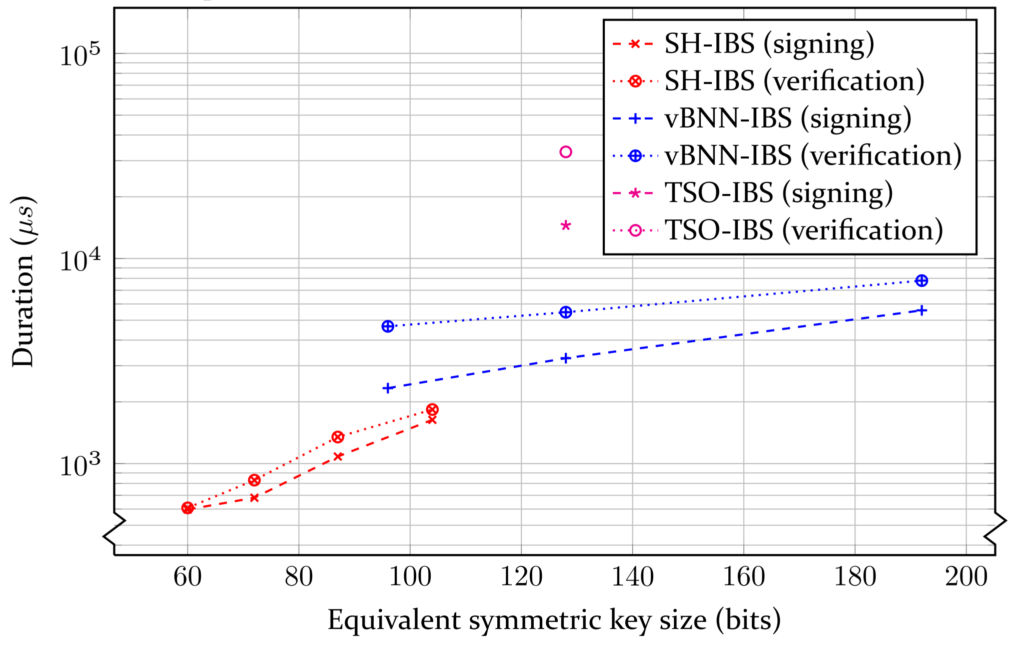


Figure 6: Comparison of signature generation and signature verification timings on the desktop platform.

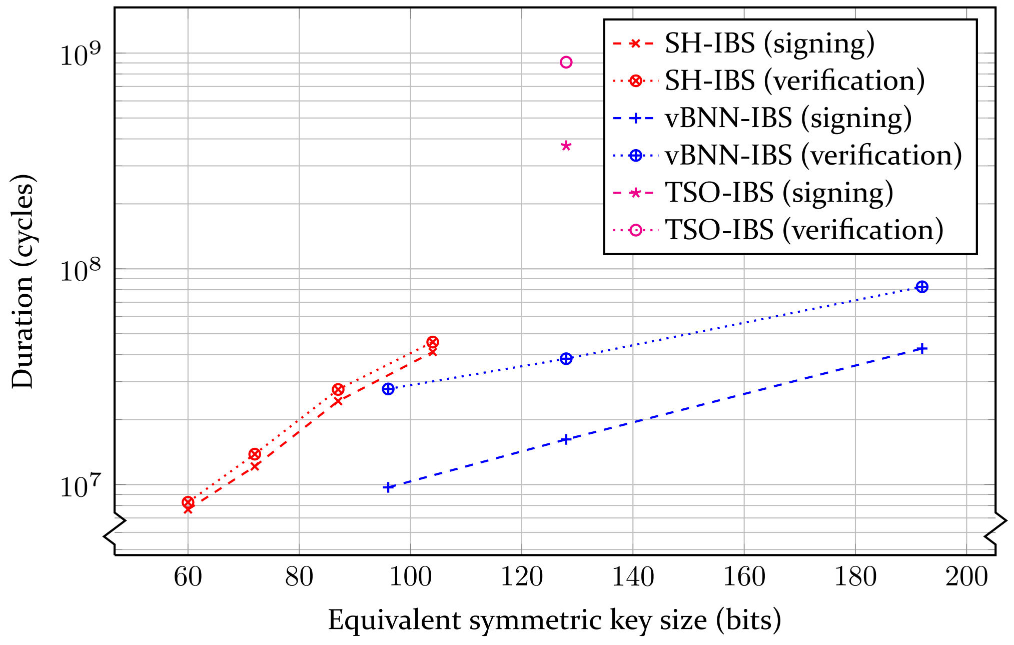


Figure 7: Cycle counts for signature generation and signature verification on the embedded platform.

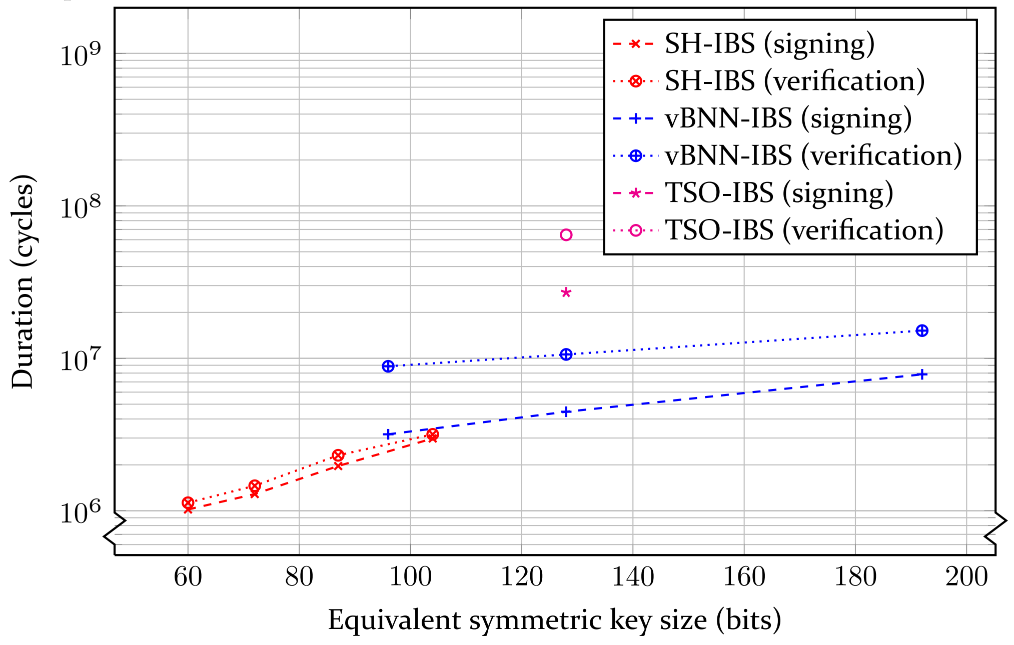


Figure 8: Cycle counts for signature generation and signature verification on the desktop platform.

the use of Edwards curves in cryptographic schemes, which lead to some performance improvements compared to classic Weierstrass curve representation [48].

There are two main directions of further research in efficient signature schemes for low power devices. One area is analyzing paring-based IBS schemes which specifically use asymmetric pairings. Currently there is no guideline describing how and when signature schemes based on symmetric parings can be securely and efficiently transformed to use asymmetric parings. This guideline would ease the development of more efficient signature schemes for constrained devices due the improved performance of asymmetric pairings.

The second option is to further analyze the application and implementation of Edwards curves. Edwards curves are currently not supported in the RELIC toolkit used in this benchmark. The implementation of Curve25519 in RELIC does group computations in the Weierstrass models without the performance advantage associated with Edwards curves. Support for Edwards curves in the RELIC toolkit would provide access to faster elliptic curve operations not only for ID-based schemes like vBNN-IBS, but also for general standard asymmetric signatures or key-exchange schemes based on elliptic curves.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Amsterdam, The Netherlands: Elsevier, 2013.

[2] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, “Efficient and Secure Source Authentication for Multicast,” in *8th Annual Internet Society Network and Distributed System Security Symposium (NDSS’01)*, Virginia, USA: ISOC, 2001, pp. 35–46.

[3] A. Shamir, “Identity-Based Cryptosystems and Signature Schemes,” in *Advances in Cryptology — CRYPTO 1984*, ser. Lecture Notes in Computer Science, G. R. Blakley and D. Chaum, Eds., vol. 196, Santa Barbara, California, USA: Springer, Aug. 1985, pp. 47–53.

[4] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, “Charm: a framework for rapidly prototyping cryptosystems,” *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, Berlin, Heidelberg, Germany: Springer, 2013.

[5] D. F. Aranha and C. P. L. Gouvêa, *RELIC is an Efficient LIbrary for Cryptography*, `http://code.google.com/p/relic-toolkit/`.

[6] E. Kiltz and G. Neven, “Identity-Based Signatures,” in *Identity-Based Cryptography*, ser. Cryptology and Information Security Series, M. Joye and G. Neven, Eds., vol. 2, Amsterdam, The Netherlands: IOS Press, 2008, pp. 31–44.

[7] D. J. Bernstein and T. Lange, *SafeCruves: choosing safe curves for elliptic-curve cryptography*, accessed 15 May 2014, 2014. [Online]. Available: `http://safecurves.cr.yp.to`.

[8] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs,” in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds., ser. Lecture Notes in Computer Science, vol. 3156, Cambridge, MA, USA: Springer, 2004, pp. 119–132.

[9] D. Boneh and M. Franklin, “Identity-Based Encryption from the Weil Pairing,” in *Advances in Cryptology — CRYPTO 2001*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139, Berlin, Heidelberg, Germany: Springer, 2001, pp. 213–229.

[10] A. Kate and I. Goldberg, “Distributed Private-Key Generators for Identity-Based Cryptography,” in *Security and Cryptography for Networks*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Prisco, Eds., vol. 6280, Berlin, Heidelberg, Germany: Springer, 2010, pp. 436–453.

[11] S. S. Al-Riyami and K. G. Paterson, “Certificateless Public Key Cryptography,” in *Advances in Cryptology — ASIACRYPT 2003*, ser. Lecture Notes in Computer Science, C.-S. Laih, Ed., vol. 2894, Berlin, Heidelberg, Germany: Springer, 2003, pp. 452–473.

[12] Common Vulnerabilities and Exposures, *CVE-2014-0160*, `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`, 2014.

[13] R. Housley, W. Polk, W. Ford, and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” IETF, RFC 3280, Apr. 2002.

[14] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP,” IETF, RFC 2560, Jun. 1999.

[15] D. Eastlake, “Transport Layer Security (TLS) Extensions: Extension Definitions,” IETF, RFC 6066, Jan. 2011.

[16] K. G. Paterson and G. Price, “A comparison between traditional public key infrastructures and identity-based cryptography,” *Information Security Technical Report*, vol. 8, no. 3, pp. 57–72, Amsterdam, Netherlands: Elsevier, 2003.

[17] A. Boldyreva, V. Goyal, and V. Kumar, “Identity-based Encryption with Efficient Revocation,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. Computer and Communications Security 2008, Alexandria, Virginia, USA: ACM, 2008, pp. 417–426.

[18] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd ed. Cambridge, UK: Cambridge University Press, 2009. [Online]. Available: `http://shoup.net/ntb/ntb-v2.pdf`.

[19] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, Florida, USA: CRC Press, 1996.

[20] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, ser. Discrete Mathematics and Its Applications. Abingdon, United Kingdom: Taylor & Francis, 2005.

[21] V. S. Miller, “Use of Elliptic Curves in Cryptography,” in *Advances in Cryptology — CRYPTO 1985*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., vol. 218, Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 1986, pp. 417–426.

[22] N. Koblitz, “Elliptic Curve Cryptosystems,” *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Providence, RI, USA: American Mathematical Society, 1987.

[23] R. L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, New York, NY, USA: ACM, Feb. 1978.

[24] NIST, “Digital Signature Standard,” Federal Information Processing Standards 186–4, Jul. 2013.

[25] J. M. Pollard, “Monte Carlo methods for index computation $(\mod p)$,” *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, Providence, RI, USA: American Mathematical Society, Jul. 1978.

[26] ECRYPT II, “ECRYPT II Yearly Report on Algorithms and Keysizes,” European Network of Excellence in Cryptology II, Tech. Rep., Sep. 2012, `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`.

[27] W. A. Stein *et al.*, *Sage Mathematics Software (Version 6.1.1)*, `http://www.sagemath.org`, 2014.

[28] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, Berlin, Heidelberg, Germany: Springer, 2001.

[29] N. P. Smart, "A Comparison of Different Finite Fields for Elliptic Curve Cryptosystems," *Computers & Mathematics with Applications*, vol. 42, no. 1–2, pp. 91–100, Amsterdam, The Netherlands: Elsevier, 2001.

[30] A. Menezes, T. Okamoto, and S. Vanstone, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field," *Information Theory, IEEE Transactions on*, vol. 39, no. 5, pp. 1639–1646, Piscataway, NJ, USA: IEEE, Sep. 1993.

[31] P. Gaudry, F. Hess, and N. P. Smart, "Constructive and Destructive Facets of Weil Descent on Elliptic Curves," *Journal of Cryptology*, vol. 15, no. 1, pp. 19–46, Berlin, Heidelberg, Germany: Springer, 2002.

[32] A. H. Koblitz, N. Koblitz, and A. Menezes, "Elliptic Curve Cryptography: The Serpentine Course of a Paradigm Shift," Cryptology ePrint Archive, Tech. Rep. Report 2008/390, 2008, `http://eprint.iacr.org/`.

[33] ECC Brainpool, "ECC Brainpool Standard Curves and Curve Generation," ECC Brainpool, Bonn, Germany, Tech. Rep. v. 1.0, Oct. 2005, `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`.

[34] H. M. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, Providence, RI, USA: American Mathematical Society, 2007.

[35] D. J. Bernstein and T. Lange, "Faster Addition and Doubling on Elliptic Curves," in *Advances in Cryptology — ASIACRYPT 2007*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., vol. 4833, Berlin, Heidelberg, Germany: Springer, 2007, pp. 29–50.

[36] A. Joux, "A One Round Protocol for Tripartite Diffie—Hellman," in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, W. Bosma, Ed., vol. 1838, Berlin, Heidelberg, Germany: Springer, 2000, pp. 385–393.

[37] ——, "A One Round Protocol for Tripartite Diffie—Hellman," *Journal of Cryptology*, vol. 17, no. 4, pp. 263–276, Berlin, Heidelberg, Germany: Springer, 2004.

[38] V. S. Miller, "Short programs for functions on Curves," *Unpublished manuscript*, 1986, `http://crypto.stanford.edu/miller/`.

[39] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems," in *Advances in Cryptology — CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442, Berlin, Heidelberg, Germany: Springer, 2002, pp. 354–369.

[40] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, Amsterdam, Netherlands: Elsevier, 2008.

[41] P. S. L. M. Barreto and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order," Cryptology ePrint Archive, Tech. Rep. Report 2005/133, 2005, `http://eprint.iacr.org/`.

[42] S. D. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate Pairing," in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, C. Fieker and D. R. Kohel, Eds., vol. 2369, Berlin, Heidelberg, Germany: Springer, 2002, pp. 324–337.

[43] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *Advances in Cryptology — ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248, Berlin, Heidelberg, Germany: Springer, 2001, pp. 514–532.

[44] X. Cao, W. Kou, L. Dang, and B. Zhao, "IMBAS: Identity-based multi-user broadcast authentication in wireless sensor networks," *Computer Communications*, vol. 31, no. 4, pp. 659–667, Amsterdam, Netherlands: Elsevier, 2008.

[45] R. Tso, C. Gu, T. Okamoto, and E. Okamoto, "Efficient ID-Based Digital Signatures with Message Recovery," in *Cryptology and Network Security*, ser. Lecture Notes in Computer Science, F. Bao, S. Ling, T. Okamoto, H. Wang, and C. Xing, Eds., vol. 4856, Berlin, Heidelberg, Germany: Springer, 2007, pp. 47–59.

[46] Committee on National Security Systems, *National information assurance policy on the use of public standards for the secure sharing of information among national security systems*, `https://www.cnss.gov/Assets/pdf/CNSSP_No%2015_minorUpdate1_Oct12012.pdf`, Oct. 2012.

[47] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: elliptic-curve points indistinguishable from uniform random strings," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ser. CCS '13, Berlin, Germany: ACM, 2013, pp. 967–980.

[48] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography - PKC 2006*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958, Berlin, Heidelberg, Germany: Springer, 2006, pp. 207–228.

[49] D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini, "A note on high-security general-purpose elliptic curves," Cryptology ePrint Archive, Tech. Rep. Report 2013/647, 2013, `http://eprint.iacr.org/`.

[50] R. Granger, T. Kleinjung, and J. Zumbrägel, "Breaking '128-bit Secure' Supersingular Binary Curves (or how to solve discrete logarithms in $\mathbb{F}_{2^{4 \cdot 1223}}$ and $\mathbb{F}_{2^{12 \cdot 367}}$)," Cryptology ePrint Archive, Tech. Rep. Report 2014/119, 2014, `http://eprint.iacr.org/`.