Accelerating V2X Cryptography through Batch Operations

Paul Bottinelli[1] and Robert Lambert[2]

[1] ISARA Corporation, Canada
paul.bottinelli@isara.com
[2] ESCRYPT Waterloo, Canada
robert.lambert@escrypt.com

Abstract. The increasing communication capabilities of vehicles are paving the way for promising road safety and traffic management applications. But the rise of connected vehicles also potentially introduces many security and privacy concerns. Thus, a vision of a successful cooperative vehicular network relies on strong security properties. Proposals such as the Security Credential Management System (SCMS) fulfil these security requirements with the concept of pseudonym certificates, relying on large-scale PKI. But since the on-board units performing these cryptographic operations are usually resource-constrained devices, it is important to consider ways to optimize and devise efficient implementations of the proposed algorithms.

In this work, we study optimizations on the mathematical and algorithmic aspects of the validation of implicit certificates and the verification of ECDSA signatures used in the SCMS. We propose efficient algorithms to validate batches of implicit certificates, providing significant savings compared to the sequential validation of the individual certificates. We also propose optimizations to the verification of ECDSA signatures when the verification is performed with an implicit certificate. Although we focus our work on the SCMS and V2X communications, our contributions are more general and apply to every system combining ECQV and ECDSA.

Keywords: Implicit certificates · ECQV · Batch ECDSA · V2X · SCMS

1 Introduction

In recent years, the automotive industry has started to explore several developments in Intelligent Transportation Systems (ITS). For example, in January 2017, the U.S. Department of Transportation (USDOT) issued a notice of proposed rulemaking [20] to mandate that vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication technology (collectively referred to as V2X) be deployed in all newly-manufactured vehicles in the next few years. In a more recent development, the US government has issued a Notice of Request for Comments on the topic of V2X Communications [21].

In a V2X system, vehicles communicate by means of broadcasting cryptographically signed Basic Safety Messages (BSMs) to nearby participants, containing information such as their speed, position and direction, in order to accurately inform neighbouring vehicles about the sender's current position and anticipated behaviour. According to the USDOT, broadcasting BSMs is expected to eliminate up to 80% of vehicle crashes [23].

However, the introduction of V2X communication technology also raises a number of security and privacy concerns, such as large-scale vehicle tracking based on the cryptographic material used to sign BSMs or attackers broadcasting invalid data resulting in dangerous road incidents. One common way to address these challenges revolves around the concept of pseudonym certificates, which provide authentication in an anonymous manner.

For instance, this approach is adopted by the Security Credential Management System (SCMS). Developed under a cooperative agreement with the USDOT, the SCMS is one of the most promising candidates for securing vehicular communications, particularly in the United States and Canada.

When deployed, Certificate Authorities (CAs) in the SCMS will issue billions of pseudonym certificates per year (estimate based on the current proposal of issuing batches of 20 pseudonym certificates per week per vehicle), making it the largest PKI ever deployed by several orders of magnitude. In the current SCMS proposal, pseudonym certificates are implemented with ECQV certificates [9], the most common form of implicit certificates.

When receiving batches of certificates, vehicles have to validate their authenticity. However, contrary to their explicit counterpart, implicit certificates do not contain a public key and a signature, but only a reconstruction value, effectively collapsing key and signature in a single value. As such, validating a received implicit certificate is achieved by first reconstructing public and private keys, and then checking that they form a valid key pair.

Since vehicles have limited computation power, it is crucial to optimize the operations they perform. The importance of optimization also applies, and arguably even more so, to the verification of BSMs. In the SCMS proposal, after having reconstructed their certificates, vehicles use the Elliptic Curve Digital Signature Algorithm (ECDSA) [14] to sign the messages they broadcast. The current scheme proposes that BSMs be broadcasted at a frequency of 10 per second. One can imagine that in a traffic-heavy situation, vehicles will thus have to verify hundreds of digital signatures per second.

Contributions. In this paper, we study optimizations on the mathematical and algorithmic aspects of the validation of implicit certificates and the verification of ECDSA signatures used in the SCMS. We propose efficient algorithms to validate batches of implicit certificates, providing significant savings compared to the sequential validation of the individual certificates. We also propose optimizations to the verification of ECDSA signatures when the issuer of those signatures provides an implicit certificate to verify it. Although we focus our work on the

SCMS and V2X communications, our contributions are more general and apply to every system combining ECQV and ECDSA.

Related work. Optimizations related to the batch processing of cryptographic operations have been an area of significant research ever since the concept of batch verification was first proposed [19].

However, efficient batch processing of implicit certificates when combined with a signature algorithm has not been studied as extensively, perhaps because no real-world system mandated their use to this day.

For instance, in [22], the author looks only at combining one signature verification and key reconstruction, and does not look at the batch processing of multiple signature verifications.

However, with the growing interest around the SCMS proposal, the topic of algorithmic improvements and optimizations of these two primitives has started to spark more attention.

In [17], the authors propose optimizations to signature verification when the chain of trust in a system is composed of a chain of shared intermediate implicit certificates.

More recently, a modification to the implicit certification scheme used in the SCMS was proposed [4], in order to make the key reconstruction operations more efficient. In this paper, we show that significant improvements can also be obtained with the primitives used in the current SCMS proposal.

Organization of the paper. In Section 2 we set up the notation we use in the remainder of the paper and present the algorithms used in the SCMS for which we propose optimizations, while in Section 3 we briefly discuss the relevant components and protocols of the SCMS. Our first contribution is presented in Section 4, where we propose performance optimizations for the validation of batches of implicit certificates. In Section 5 we present our second contribution, consisting of performance optimizations for the verification of batches of ECDSA signatures being verified with an implicit certificate. We implemented the proposed algorithms and give some concrete timings and speedup factors in Section 6. The concluding Section 7 highlights our contributions and some potential future research directions.

2 Preliminaries

Notation

Let p be a prime power and let E be an elliptic curve defined over the finite field $\mathbb{F}_p$. Let the generator of the group of points on the elliptic curve $E(\mathbb{F}_p)$ be denoted by G, and let q be the prime order of G. For simplicity, we sometimes omit the reduction modulo q in the algorithms presented in this paper.

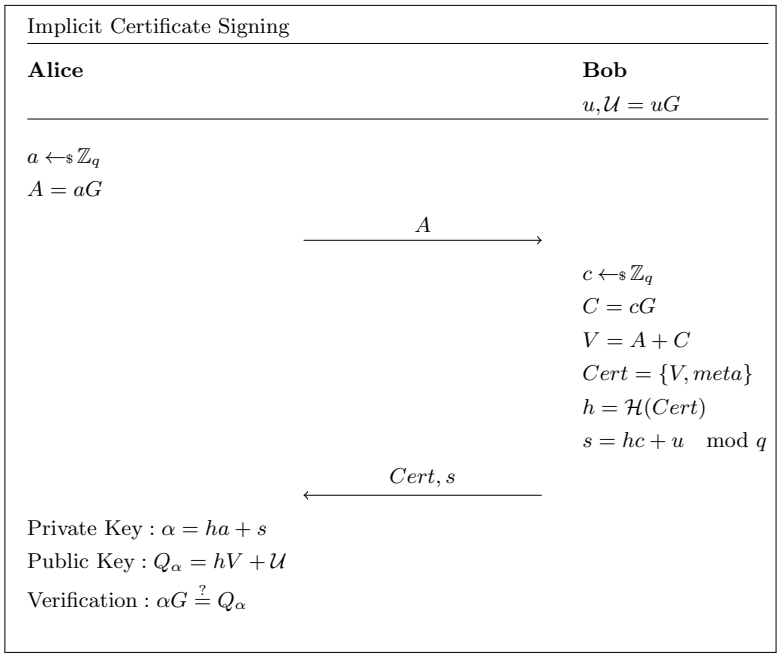

Fig. 1. ECQV

We denote by $x \leftarrow_{\$} \mathbb{Z}_q$ the action of randomly sampling an element x from $\mathbb{Z}_q$. The notation $a \stackrel{?}{=} b$ represents the procedure by which one checks whether the quantities a and b are equal.

ECQV

The SCMS proposal mandates the use of implicit certificates for V2X communication. Contrary to more traditional forms of certificates, implicit certificates save space by combining the public key and the signature into a single value, called the reconstruction value.

In order to verify a message signed by the owner of this certificate, the recipient first has to reconstruct the associated public key by combining the CA's public key and the reconstruction value. The SCMS uses perhaps the most common form of implicit certificates: Elliptic Curve Qu-Vanstone (ECQV) certificates [9].

Figure 1 describes the procedure by which an ECQV Implicit Certificate is issued by Bob to Alice, where $u, \mathcal{U}$ is the issuer key pair, V is the public key reconstruction value, s is the private key reconstruction value, $\mathcal{H}$ is a collision-resistant hash function and *meta* corresponds to some metadata associated with the certificate, such as the validity period or some identification information for the subject or the issuer.

ECDSA

Additionally, the SCMS proposal mandates the use of the Elliptic Curve Digital Signature Algorithm (ECDSA) [14], standardized in [1,11] as the signature algorithm for V2X communication.

We present the signing and verification algorithms of ECDSA in Figure 2, where the function $\phi(R)$ returns the x-coordinate of the elliptic curve point R.

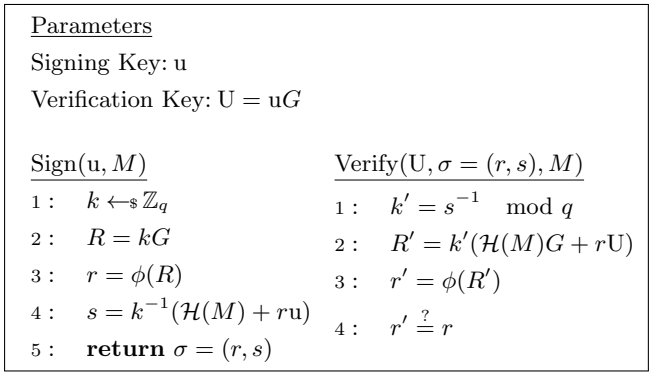

Fig. 2. ECDSA Signing and Verification algorithms

3 The Security Credential Management System

This section describes some of the general ideas behind the Security Credential Management System (SCMS). The SCMS is a large and complex system, composed of more than a dozen entities covering aspects such as enrolment of entities, certificate issuance and certificate revocation. Most of these entities are not relevant to our work. Since our focus is on the pseudonym certificate provisioning process, namely the batch certificate reconstruction, and the verification of the signatures on BSMs, we only describe the entities relevant for these processes, for the sake of simplicity. A detailed description of the complete proposed system can be found in [8].

3.1 Relevant Entities in the SCMS

The following entities take part in the pseudonym certificate provisioning process and in the signing and verifying of BSMs.

- *End-Entities* (EEs), be they in-vehicle On-Board Equipment (OBE) or Road-Side Equipment (RSE) located in the traffic infrastructure, are the actual participants in V2X communications. They are the recipients of implicit certificates and they send and verify signed BSMs.

– The *Registration Authority* (RA) is the main point of contact for EEs. It validates and processes their requests for pseudonym certificates before forwarding them to the PCA, and forwards the batches of certificates issued back to the EEs.
– The *Pseudonym Certificate Authority* (PCA) is in charge of issuing short-lived pseudonym certificates for devices.

3.2 Implicit Certificate Provisioning

Figure 3 describes the implicit certificate provisioning process proposed in the SCMS, where implicit certificates are ECQV certificates as defined in [9]. Note that we also sometimes refer to these implicit certificates as pseudonym certificates, since this is the name they are given in the SCMS.

The certificate provisioning process in the SCMS includes a proposal called the butterfly key expansion process, which is used to efficiently implement the pseudonym certificate concept in the SCMS. More specifically, it allows EEs to obtain a large number of pseudonym certificates, anonymous to the PCA and RA, with a single short request message. In Figure 3, the function $f_k()$ is known as the expansion function, and is a pseudo-random function parameterized by a unique secret key k of length m bits shared only between the EE and the RA; $u, \mathcal{U}$ is the CA key pair, and $\mathcal{H}$ is a suitable hash function.

Upon reception of a batch of certificates, EEs have to validate the certificates that the batch contains. Validation of an implicit certificate essentially achieves two purposes: first, it assures the EE that it was indeed the PCA that issued the certificate and second, it ensures that the value certified was the one submitted: which, as a result, ensures that the public reconstructed key corresponds to the private reconstructed key and that other participants will thus be able to verify signatures on messages.

To further illustrate the necessity of certificate validation, we refer the reader to the standard [9, Section 3.6], where the Section *Processing the Response to a Cert_Request: Cert_Reception* requires that this validation step be performed upon reception of an implicit certificate issued by the CA.

An additional and more pragmatic point in favour of batch validation is the fact that the presence of one invalid certificate may mean that the RA, or worse, the PCA, was compromised. In this case, knowing precisely which certificate was invalid might be a secondary concern, and instead the whole batch should be dropped as a precaution.

3.3 Signed BSM Broadcasting in the SCMS

After having validated and reconstructed their pseudonym certificates, EEs may participate in V2X communication by signing and broadcasting Basic Safety Messages (BSMs). BSMs are broadcasted upwards of 10 times per second. To sign a message M, an EE uses its private, reconstructed key, α, as computed in Figure 3, to sign a message using ECDSA, $Sign(\alpha, M) = \sigma_M$.

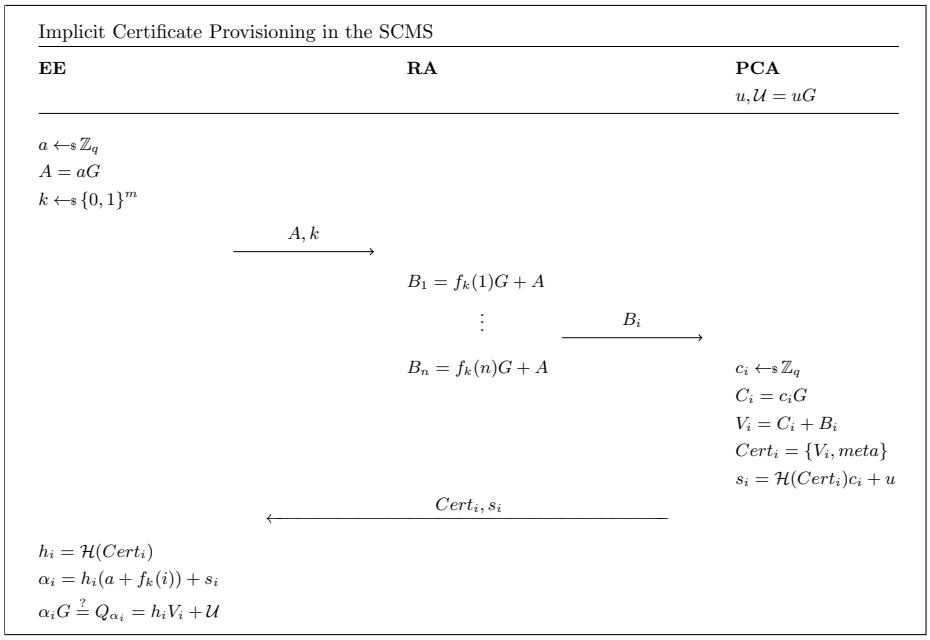

Fig. 3. Implicit Certificate Provisioning in the SCMS

The EE then broadcasts the message M, together with its signature σ_M and the corresponding certificate, Cert, containing the public information associated with the private key that the EE used for signing M.

In order to verify the signature σ_M, any entity receiving must first reconstruct the corresponding public key. This is done by first hashing the certificate $h = \mathcal{H}(\mathsf{Cert})$, then computing the public key Q_α by taking the public key reconstruction value included in the certificate V and computing $Q_\alpha = hV + \mathcal{U}$.

In the following sections, we look at various methods to optimize these operations.

4 Batch Validation of Implicit Certificates

As already discussed in 3.2 and since the validation of implicit certificates is mandated by the standard [9], it is interesting to look at ways this computation can be optimized.

Let us consider a batch of n implicit certificates, $Cert_1, \ldots, Cert_n$ received by an end-entity. The EE has to validate the certificates received by checking that:

$$\alpha_i G \stackrel{?}{=} Q_{\alpha_i} = h_i V_i + \mathcal{U} \text{ for } 1 \leq i \leq n.$$

This requires two elliptic curve scalar multiplications and one elliptic curve scalar addition. To this we must add the two integer additions and the integer

multiplication required to compute

$$\alpha_i = h_i(a + f_k(i)) + s_i.$$

By naively validating each of the n certificates independently, the EE must compute $2n$ elliptic curve scalar multiplications, n elliptic curve scalar additions as well as n integer multiplications and $2n$ integer additions.

Computationally, the most expensive operation performed when validating implicit certificates is the elliptic curve scalar multiplication. We therefore propose a method to reduce the amount of such scalar multiplications by validating implicit certificates in batches.
An individual implicit certificate validation is:

$$(h_i(a + f_k(i)) + s_i)G = h_i V_i + \mathcal{U}.$$

Inverting h_i and multiplying through leaves the V_i with unit multiples:

$$(a + f_k(i) + h_i^{-1} s_i)G = V_i + h_i^{-1}\mathcal{U}.$$

Summing n validations together yields:

$$\left(\sum_{i=1}^{n} a + f_k(i) + h_i^{-1} s_i\right) G = \left(\sum_{i=1}^{n} h_i^{-1}\right)\mathcal{U} + \sum_{i=1}^{n} V_i.$$

With this transformation, the batch certificate validation can be performed with two elliptic-curve point multiplications (and these of points known well in advance) and a sum of elliptic curve points.

In order to obtain the inverse values h_i^{-1}, another batch operation is useful, that is simultaneous inversion (also known as Montgomery's trick) [18]. First, start by setting $m_1 = h_1$, $m_2 = h_1 h_2$, ..., $m_n = h_1 h_2 \cdots h_n$, and then invert m_n. We can thus obtain the inverse of the last element by computing $h_n^{-1} = m_{n-1} m_n^{-1}$. Subsequently, we can obtain $m_{n-1}^{-1} = m_n^{-1} h_n$, which brings us to where we would be with only the first $n-1$ values for h_i, allowing us to obtain successively all the inverses with $3(n-1)$ finite field multiplications and a single inverse.

Another important technique we will employ is Shamir's trick [12], which is used to efficiently compute the quantity $aP+bQ$, for scalars a, b and elliptic curve points P, Q. More specifically, the elliptic curve doubles needed to compute the linear combination can be shared, compared with the two scalar multiplications and one point addition needed were we to do it naively.

What performance improvement can be had by batching the certificate verification? First we move the elliptic curve point multiplications to one side, since it is possible to compute them simultaneously and save elliptic curve point doublings:

$$\left(\sum_{i=1}^{n} a + f_k(i) + h_i^{-1} s_i\right) G + \left(\sum_{i=1}^{n} -h_i^{-1}\right)\mathcal{U} = \sum_{i=1}^{n} V_i. \tag{1}$$

The points G and $\mathcal{U}$ are long-term and can be supplied with tables to improve performance. Consider a minimal scheme for tables on G and $\mathcal{U}$ consisting of the extra values $G_{\text{hi}} = 2^{\lceil t/2 \rceil} G$ and $\mathcal{U}_{\text{hi}} = 2^{\lceil t/2 \rceil} \mathcal{U}$, where t is the length of the point multipliers (the points scaled by 2 to the power halfway up the length of the multiplier). This precomputed value reduces the number of doubles required to compute the linear combination, breaking down a multiple

$$\gamma G = \gamma_{\text{hi}} G + \gamma_{\text{lo}} G,$$

where γ_{hi} and γ_{lo} are half-size multiples, which are just the high and low bits of γ divided at $\lceil t/2 \rceil$. If this linear combination is performed with a bitwise left-to-right add and accumulator double, then the number of accumulator doubles is halved. The same transformation can be made for a multiple of $\mathcal{U}$. In this way, a linear combination of G and $\mathcal{U}$ can also be performed, saving half of the accumulator doubles.

With this precomputation, what is the cost of computing Eq. 1? In the US SCMS system, the NIST P256 curve is specified. This curve is often implemented with Jacobian coordinates. From the explicit formula database [7], each double requires four finite field multiplies and five squares: $4M + 5S$. Assuming squares are implemented as multiplication, this is $9M$. Each addition requires $10M + 3S$, or roughly $13M$. Assume that we have the scaled values of G and $\mathcal{U}$ allowing us to save half the doubles. Table entries for this curve are $2 \times 2^8/2^3 = 2^6 = 64$ bytes, so these two entries take only 128 bytes. Assume that roughly half of the bits are one, which require adds into a (point) accumulator, and that the accumulator is doubled. Then each 128-bit (half) multiplier requires on average 64 adds. The linear combination then requires four of these half multipliers to be processed, plus the accumulator doubles: $4 \times 64 \times 13M + 128 \times 9M = 4480M$. The sum of the V_i costs $n \times (13M + 3M)$ (where the $3M$ is for the batch inversion). We ignore the cost of the normalization of the point from Jacobian coordinates.

What does the naive approach cost, applying a similar level of implementation optimization? This will need n invocations of point multiply for G, V_i and an addition of $\mathcal{U}$. The V_i will not be known in advance, requiring 256 doubles and the G and V_i multiples will each require 128 adds on average. This is $256 \times 13M + 256 \times 9M = 5632M$. This needs to be computed n times.

One way of comparing these alternatives is to state that the linear combination of the batched version is already cheaper than a single naive validation, and that adding certificates adds only $16M$ for the batched version but $5632M$ for the naive version, or about 350 times less incremental cost for each new certificate.

5 Batch Verification of ECDSA Signatures from Implicit Certificates

In this section, we propose optimizations to the verification of ECDSA signatures when verified with implicit certificates.

When verifying a batch of signatures, and particularly in the setting of the SCMS, messages may be signed by different private keys, whether from one signer that is rotating certificates or from different signers. We thus focus on this case, as opposed to the more traditional case where all signatures are issued by the same private key which has already been extensively covered in the literature [16,15,10].

We consider a common variant of ECDSA, known as ECDSA* [2], where the signature on a message includes the point R, and not just the x-coordinate of the point, denoted r in Figure 2. More specifically, an ECDSA signature (r, s) becomes (R, s) with ECDSA*. The first step when verifying an ECDSA* signature is thus to take the x-coordinate of the point R to obtain r, and then proceed to verify the signature by checking that the reconstructed point obtained R' is equal to the point received.

This variant was proven to be equivalent in terms of security to the standard ECDSA [2], and the standard upon which the SCMS proposal is based endorses its use as an alternative to the standard ECDSA [13].

With ECDSA*, the signature verification computation equation thus becomes:

$$R \stackrel{?}{=} s^{-1}(\mathcal{H}(M)G + r\mathrm{U}).$$

Substitute the value of the public key $\mathrm{U} = \mathcal{H}(\mathsf{Cert})V + \mathcal{U}$ to get

$$R = s^{-1}(\mathcal{H}(M)G + r(\mathcal{H}(\mathsf{Cert})V + \mathcal{U})).$$

Now, consider a batch of signatures:

$$\begin{aligned}
R_1 &= s_1^{-1}(\mathcal{H}(M_1)G + r_1(\mathcal{H}(\mathsf{Cert}_1)V_1 + \mathcal{U})),\\
R_2 &= s_2^{-1}(\mathcal{H}(M_2)G + r_2(\mathcal{H}(\mathsf{Cert}_2)V_2 + \mathcal{U})),\\
&\vdots\\
R_n &= s_n^{-1}(\mathcal{H}(M_n)G + r_n(\mathcal{H}(\mathsf{Cert}_n)V_n + \mathcal{U})).
\end{aligned}$$

Similarly to what we presented in Section 4, we can thus verify the whole batch of signatures by verifying that

$$\sum_{i=1}^{n} R_i = \sum_{i=1}^{n} s_i^{-1}(\mathcal{H}(M_i)G + r_i(\mathcal{H}(\mathsf{Cert}_i)V_i + \mathcal{U})) \tag{2}$$

At this point, it is important to note that naive batch verification as performed above is vulnerable to some specific kinds of forgeries, where an attacker might be able to generate signatures that pass the batch verification test while being invalid on their own.

This was highlighted in [6], where the authors presented two attacks against a batch verification scheme similar to Eq. 2, proposed in [16]. In Appendix A, we show an attack that applies to the naive Eq. 2 batch verification proposal.

To protect against such attacks, we introduce the notion of a randomizer ρ_i, initially proposed in [19]. This random integer is used to multiply both sides of the batch verification equation and renders the attacks described in [6] and A infeasible.

Thus, with randomizing multiplier ρ_i, the relation to be checked for a single ECDSA validation is:

$$\begin{aligned}\rho_i R_i &= \rho_i s_i^{-1} \left(\mathcal{H}(M_i)G + r_i(\mathcal{H}(\mathsf{Cert}_i)V_i + \mathcal{U})\right) \\ &= \rho_i s_i^{-1}\mathcal{H}(M_i)G + \rho_i s_i^{-1} r_i \mathcal{H}(\mathsf{Cert}_i)V_i + \rho_i s_i^{-1} r_i \mathcal{U}.\end{aligned}$$

Set $\beta_i = \rho_i s_i^{-1} r_i \mathcal{H}(\mathsf{Cert}_i)$, and multiply through by β_i^{-1} to obtain:

$$\beta_i^{-1}\rho_i R_i = \beta_i^{-1}\rho_i s_i^{-1}\mathcal{H}(M_i)G + V_i + \beta_i^{-1}\rho_i s_i^{-1} r_i \mathcal{U}.$$

Again β_i^{-1} can be calculated in a batch, with only one inversion, using Montgomery's trick as already presented in Section 4.

This form will yield a batch verification system where the V_i need only be added together:

$$\sum_{i=1}^{n} \beta_i^{-1}\rho_i R_i = \left(\sum_{i=1}^{n} \beta_i^{-1}\rho_i s_i^{-1}\mathcal{H}(M_i)\right) G + \sum_{i=1}^{n} V_i + \left(\sum_{i=1}^{n} \beta_i^{-1}\rho_i s_i^{-1} r_i\right) \mathcal{U}.$$

The other elliptic curve point which is not known long-term is R_i. If we want to balance the multiple of V_i and R_i, this can be done in the same way as was done in [2].

More concretely, set $\gamma_i = \beta_i^{-1}\rho_i R_i$ and write $\gamma_i = \nu_i/\delta_i$, where ν_i and δ_i are (almost) half length values. Then multiply through by δ_i:

$$\nu_i R_i = \delta_i\beta_i^{-1}\rho_i s_i^{-1}\mathcal{H}(M_i)G + \delta_i V_i + \delta_i\beta_i^{-1}\rho_i s_i^{-1} r_i \mathcal{U}. \tag{3}$$

This yields a batch validation with half-length multiples for each elliptic curve point not known long-term. This will allow half of the doubles to be removed in the simultaneous calculation of 3 as a whole:

$$\sum_{i=1}^{n} \nu_i R_i = \left(\sum_{i=1}^{n} \delta_i\beta_i^{-1}\rho_i s_i^{-1}\mathcal{H}(M_i)\right) G + \sum_{i=1}^{n} \delta_i V_i + \left(\sum_{i=1}^{n} \delta_i\beta_i^{-1}\rho_i s_i^{-1} r_i\right) \mathcal{U}. \tag{4}$$

Let's make a rough performance evaluation along the lines of Section 4 above. Since the multiples of previously unknown points are half length, at least 128 doubles are required. Assuming that on average half of the bits of the two half-length multiples are one, then we need $256/2 = 128$ additions as well. Together, this is $128 \times 9M + 128 \times 13M = 2816M$ for each signature, for an incremental

cost of $2816M$. The multiples of G and $\mathcal{U}$, if we assume one scaled value each, half-way up (as in [2]), and assume one-half one bits on average, will require 256 additions, but no extra doubles, which can be shared with the half-length multiples. This is a base cost of $256M = 3328M$.

The naive verification will take one linear combination of known points, plus a multiple of an unknown point, or with the same assumptions $256 \times 13M + 128 \times 13M + 256 \times 9M = 7296M$ per signature. This is more than twice the incremental cost of the batch verification.

Note on different implicit certificate issuers. Finally, note that up to this point we have considered the case where the certificate issuer was the same for all implicit certificates. While this is likely in practice, it may be the case that some signatures an EE must verify are from certificates issued by different PCAs. It is easy to see from Eq. 4 that the optimizations we propose could also be applied to a situation where different $\mathcal{U}$s are used, even though the performance gain would decrease for every new CA.

6 Performance Evaluation

Table 1 summarizes the performance of the batch ECQV certificate validation and of the batch ECDSA verification presented in Sections 4 and 5, respectively.

Algorithm	Performance	Example ($n = 20$)
Naive Batch ECQV validation	$5683n$	113660
Efficient Batch ECQV validation	$16n + 4480$	4800 ($\sim 23.7\times$ faster)
Naive Batch ECDSA verification	$7296n$	145920
Efficient Batch ECDSA verification	$2816n + 3328$	59648 ($\sim 2.4\times$ faster)

Table 1. Performance comparison of the batch algorithms presented above. The performance is measured in finite field multiplications M. The value of n used in the example corresponds to the currently proposed size of a batch of implicit certificates, and also represents a reasonable amount of signatures to be verified simultaneously by a vehicle driving in moderate traffic.

To confirm our findings related to performance optimization, we implemented the proposed algorithms using the RELIC toolkit [3] for benchmarking purposes. We implemented both our implicit certificate batch validation algorithm and our efficient batch ECDSA verification.

The benchmarks were run using RELIC's default settings, 10k times, on an Intel Core i7 processor running at 2.2 GHz. The results are presented in Table 2. We see that we obtain speedup factors comparable to the ones obtained through our rough performance evaluation presented in the later parts of Sections 4 and 5, respectively.

Algorithm	$n = 20$	Speedup factor
Naive Batch ECQV validation	13963 μs	$\sim 23.91\times$ faster
Efficient Batch ECQV validation	584 μs	
Naive Batch ECDSA verification	33059 μs	$\sim 2.34\times$ faster
Efficient Batch ECDSA verification	14114 μs	

Table 2. Benchmarks of our algorithms, implemented using the RELIC toolkit [3]. Results are reported in microseconds, for a batch size of 20.

7 Conclusion

In this paper, we presented several methods to speed up operations performed by end-entities in the SCMS. We proposed optimizations for the validation of batches of implicit certificates, allowing EEs to efficiently determine whether a batch of pseudonym certificates is valid or not.

We also proposed optimizations to the verification of ECDSA signatures when verification is performed with an implicit certificate. Since the computation power of EEs might be limited, these optimizations may have a significant impact on the number of operations that EEs can perform, thereby increasing the security of the system as a whole.

Finally, we implemented the algorithms presented in this paper to support our performance claims. We obtained timings and speedup factors comparable to the expected performance devised through our rough performance evaluation.

In terms of future work, an interesting avenue would be to look at the different tradeoffs between speed and size of the precomputated tables for efficient elliptic curve arithmetic. This is of particular interest in memory-constrained devices that might have limited storage available.

Additionally, proving the security of our batch implicit certificate validation algorithm might be a worthwhile direction for future development. However, this would probably be a significant undertaking, since it would require to start by clearly defining a security model encompassing implicit certificates, the butterfly key expansion process in the SCMS and the batched operations we propose.

References

1. ANSI X9.62, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) (September 1998), in: American National Standards Institute, X9-Financial Services
2. Antipa, A., Brown, D.R.L., Gallant, R.P., Lambert, R.J., Struik, R., Vanstone, S.A.: Accelerated verification of ECDSA signatures. In: Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers. pp. 307–318 (2005). https://doi.org/10.1007/11693383_21, `https://doi.org/10.1007/11693383_21`

3. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic
4. Barreto, P.S.L.M., Jr., M.A.S., Ricardini, J.E., Patil, H.K.: Schnorr-based implicit certification: improving the security and efficiency of v2x communications. Cryptology ePrint Archive, Report 2019/157 (2019), https://eprint.iacr.org/2019/157
5. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (ed.) Advances in Cryptology — EUROCRYPT'98. pp. 236–250. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
6. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.J.: Faster batch forgery identification. Cryptology ePrint Archive, Report 2012/549 (2012), https://eprint.iacr.org/2012/549
7. Bernstein, D.J., Lange, T.: Explicit formulas database. https://hyperelliptic.org/EFD, online; accessed 3-May-2019
8. Brecht, B., Therriault, D., Weimerskirch, A., Whyte, W., Kumar, V., Hehn, T., Goudy, R.: A Security Credential Management System for V2X Communications. CoRR **abs/1802.05323** (2018), http://arxiv.org/abs/1802.05323
9. Campagna, M.: Standards for efficient cryptography, SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV). Tech. rep., Certicom Research (2013)
10. Cheon, J.H., Lee, M.: Improved batch verification of signatures using generalized sparse exponents. Computer Standards & Interfaces **40**, 42–52 (2015). https://doi.org/10.1016/j.csi.2014.12.004, https://doi.org/10.1016/j.csi.2014.12.004
11. FIPS PUB 186-3 Federal Information Processing Standards Publication Digital Signature Standard (DSS) (June 2009), in: U.S. Department of Commerce/National Institute of Standards and Technology
12. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer Publishing Company, Incorporated, 1st edn. (2010)
13. IEEE Standard for Wireless Access in Vehicular Environments — Security Services for Applications and Management Messages. IEEE Std 1609.2-2013 (Revision of IEEE Std 1609.2-2006) pp. 1–289 (April 2013). https://doi.org/10.1109/IEEESTD.2013.6509896
14. Johnson, D., Menezes, A., Vanstone, S.: The Elliptic Curve Digital Signature Algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (Aug 2001). https://doi.org/10.1007/s102070100002, http://dx.doi.org/10.1007/s102070100002
15. Karati, S., Das, A., Chowdhury, D.R., Bellur, B., Bhattacharya, D., Iyer, A.: New algorithms for batch verification of standard ECDSA signatures. J. Cryptographic Engineering **4**(4), 237–258 (2014). https://doi.org/10.1007/s13389-014-0082-x, https://doi.org/10.1007/s13389-014-0082-x
16. Karati, S., Das, A., Roychowdhury, D., Bellur, B., Bhattacharya, D., Iyer, A.: Batch verification of ecdsa signatures. In: Mitrokotsa, A., Vaudenay, S. (eds.) Progress in Cryptology - AFRICACRYPT 2012. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
17. Kwon, H.Y., Lee, M.K.: Fast signature verification with shared implicit certificates for vehicular communication. In: Barolli, L., Xhafa, F., Yim, K. (eds.) Advances on Broad-Band Wireless Computing, Communication and Applications. Springer International Publishing (2017)

18. L. Montgomery, P.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of Computation - Math. Comput. **48**, 243–243 (01 1987). https://doi.org/10.1090/S0025-5718-1987-0866113-7
19. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? — Complexity trade-offs with the digital signature standard —. In: De Santis, A. (ed.) Advances in Cryptology — EUROCRYPT'94. pp. 77–85. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
20. NHTSA-2016-0126: Notice of proposed rulemaking (NPRM) on V2V communications. `http://www.safercar.gov/v2v/pdf/V2V%20NPRM_Web_Version.pdf` (January 2017), online; accessed 3-May-2019
21. Notice of Request for Comments: V2X Communications. `https://www.federalregister.gov/documents/2018/12/26/2018-27785/notice-of-request-for-comments-v2x-communications` (December 2018), online; accessed 3-May-2019
22. Struik, R.: Batch Computations Revisited: Combining Key Computations and Batch Verifications. In: Proceedings of the 17th International Conference on Selected Areas in Cryptography. pp. 130–142. SAC'10, Springer-Verlag, Berlin, Heidelberg (2011), `http://dl.acm.org/citation.cfm?id=1964441.1964454`
23. U.S. Department of Transportation (USDOT): How connected vehicles work. `https://www.its.dot.gov/factsheets/pdf/JPO_HowCVWork_v3.pdf`, [Online]

A Attack on Naive Batch Verification

Recall ECDSA verify:

$$R' = s^{-1}(\mathcal{H}(M)G + rQ).$$

A simple path to forgery is as follows: Pick $\rho_1, \rho_2 \in \mathbb{Z}_q$, compute $R_1 := \rho_1 G$ and $R_2 := \rho_2 G$ and derive r_1, r_2 as the integer representation of the x-coordinates of R_1, resp., R_2, namely $r_1 = \phi(R_1)$ and $r_2 = \phi(R_2)$. Then set

$$s_1 = \frac{(r_2/r_1)\mathcal{H}(M_1) - \mathcal{H}(M_2)}{(r_2/r_1)(\rho_1 + \rho_2)} = \frac{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}{r_2(\rho_1 + \rho_2)}$$

and

$$\begin{aligned} s_2 &= -s_1\Big(\frac{r_2}{r_1}\Big) = -\frac{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}{r_2(\rho_1 + \rho_2)} \cdot \frac{r_2}{r_1} \\ &= -\frac{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}{r_1(\rho_1 + \rho_2)} \end{aligned}$$

We verify that a naive batch verification succeeds.

$$\begin{aligned} R_1 + R_2 &= s_1^{-1}(\mathcal{H}(M_1)G + r_1Q) + s_2^{-1}(\mathcal{H}(M_2)G + r_2Q) \\ &= \left(\frac{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}{r_2(\rho_1 + \rho_2)}\right)^{-1}(\mathcal{H}(M_1)G + r_1Q) + \\ &\qquad \Big(-\frac{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}{r_1(\rho_1 + \rho_2)}\Big)^{-1}(\mathcal{H}(M_2)G + r_2Q) \\ &= \left(\frac{r_2(\rho_1 + \rho_2)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}\right)(\mathcal{H}(M_1)G + r_1Q) - \\ &\qquad \Big(\frac{r_1(\rho_1 + \rho_2)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)}\Big)(\mathcal{H}(M_2)G + r_2Q) \\ &= \frac{r_2(\rho_1 + \rho_2)(\mathcal{H}(M_1)G + r_1Q)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} - \frac{r_1(\rho_1 + \rho_2)(\mathcal{H}(M_2)G + r_2Q)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} \\ &= \frac{r_2(\rho_1 + \rho_2)(\mathcal{H}(M_1)G + r_1Q) - r_1(\rho_1 + \rho_2)(\mathcal{H}(M_2)G + r_2Q)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} \\ &= \frac{(\rho_1 + \rho_2)\Big(r_2(\mathcal{H}(M_1)G + r_1Q) - r_1(\mathcal{H}(M_2)G + r_2Q)\Big)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} \\ &= \frac{(\rho_1 + \rho_2)\Big(r_2\mathcal{H}(M_1)G - r_1\mathcal{H}(M_2)G\Big)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} \\ &= \frac{(\rho_1 + \rho_2)G\Big(r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)\Big)}{r_2\mathcal{H}(M_1) - r_1\mathcal{H}(M_2)} \\ &= (\rho_1 + \rho_2)G \\ &= R_1 + R_2 \end{aligned}$$