Batch Computations Revisited: Combining Key Computations and Batch Verifications

René Struik

723 Carlaw Ave, Toronto ON M4K 3K8, Canada
rstruik.ext@gmail.com

Abstract. We consider the effect of combining the key computation step in particular key agreement protocols, such as ECMQV and static-DH, with verifying particular elliptic curve equations, such as those related to ECDSA signature verification. In particular, we show that one can securely combine ECDSA signature verification and ECMQV and static-ECDH key computations, resulting in significant performance improvements, due to saving on doubling operations and exploiting multiple point multiplication strategies. Rough estimates (for non-Koblitz curves) suggest that the incremental cost of ECDSA signature verification, when combined with ECDH key agreement, improves by a factor 2.3× compared to performing the ECDSA signature verification separately and by a factor 1.7×, when the latter is computed using the accelerated ECDSA signature verification technique described in [3]. Moreover, the total cost of combined ECDSA signature verification and ECDH key agreement improves by 1.4×, when compared to performing these computations separately (and by 1.2×, if accelerated ECDSA signature verification techniques are used). This challenges the conventional wisdom that with ECC-based signature schemes, signature verification is always considerably slower than signature generation and slower than RSA signature verification. These results suggest that the efficiency advantage one once enjoyed using RSA-based certificates with ECC-based key agreement schemes may be no more: one might as well use an ECC-only scheme using ECDSA-based certificates. Results apply to all prime curves standardized by NIST, the NSA 'Suite B' curves, and the so-called Brainpool curves.

Keywords: elliptic curve, authenticated key agreement, key computation, certificate verification, ECDSA, efficient computations.

1 Introduction

Discrete logarithm based authenticated public-key key agreement protocols typically include the following steps:

– *Key contributions.* Each party randomly generates a short-term (ephemeral) public key pair and communicates the ephemeral public key to the other party (but not the private key). In addition, it may communicate its long-term static public key.

A. Biryukov, G. Gong, and D.R. Stinson (Eds.): SAC 2010, LNCS 6544, pp. 130–142, 2011.

- *Key establishment.* Each party computes the shared key based on the static and ephemeral public keys it obtained from the other party and based on the static and ephemeral private keys it generated itself. The key computation is such that either party indeed arrives at the same shared key.
- *Key authentication.* Each party verifies the authenticity of the long-term static key of the other party, to obtain evidence that the only party that may have been capable of computing the shared key is, indeed, its perceived communicating party.
- *Key confirmation.* Each party evidences possession of the shared key to the other party, usually by communicating a message authentication check value over the strings corresponding to the key contributions communicated by either party using a key derived from the shared key. This confirms to each party the true identity of the other party and proves that that party successfully computed the shared key.

The key authentication step is typically carried out separately from the other protocol steps described above and usually involves checking the validity of a certificate that vouches for the authenticity of the binding between a party and its private key (by means of the corresponding public key). While separating the key authentication step from the remaining protocol steps allows more flexibility (e.g., in use of certificates), it may present a significant computational burden, since the online computational cost of the key agreement protocol – and thereby its running time – is dominated by the cost of the key authentication step and that of the key computation performed during the key establishment step. Here, we assume key authentication has to be performed at each instantiation of the protocol.

In this paper, we consider a method for securely combining the key authentication and the key computation steps, rather than carrying these out separately, thus realizing a significant reduction of the online computational cost of the resulting protocol and, thereby, of its running time. While this approach leads to some loss of flexibility (since one has to carry out computations on similar mathematical objects), quite remarkable efficiency gains seem possible.

The remainder of the paper is organized as follows. In §2, we provide some preliminaries that facilitate the main exposition of the paper, while §3 gives applications that illustrate the performance advantages that can be reaped when applying joint computations in the context of some well-known authenticated key agreement schemes. Summary conclusions appear in §4.

2 Preliminaries

In this section, we introduce some simple notions that facilitate the exposition in the remainder of the paper. §2.1 introduces some probability results, which we apply in §2.2 to key establishment schemes. §2.3 summarizes some properties of ECDSA and the related signature scheme ECDSA*.

The exposition below is relative to a group $\mathbb{E}$ with (cyclic) subgroup $\mathbb{G}$ of prime order n generated by some point G and with identity element $\mathcal{O}$.

2.1 Probabilities

Let f be a function defined on $\mathbb{E}$. For each $z \in \mathrm{Im}(f)$, define $f^{-1}(z) := \{X \in \mathbb{E} \mid f(X) = z\}$ and define $d_\infty(f^{-1}) := \max\{|f^{-1}(z)| \mid z \in \mathrm{Im}(f)\}$.

Lemma 1. *Let $K_A \in \mathbb{E}$, let $\Delta \in \mathbb{G}$, and let $\lambda \in_R \Omega$, where $\Omega \subset {\mathbb{Z}_n}^*$. Let $K'_A := K_A + \lambda \cdot \Delta$. Then*

$$\max_{K \in \mathbb{E}} \Pr\{K'_A = K\} = \begin{cases} 1 & \text{if } \Delta = \mathcal{O}; \\ |\Omega|^{-1} & \text{otherwise.} \end{cases}$$

Proof. Trivial. □

Lemma 2. *Let $K_A \in \mathbb{E}$, let $\Delta \in \mathbb{G}$, and let $\lambda \in_R \Omega$, where $\Omega \subset {\mathbb{Z}_n}^*$. Let $K'_A := K_A + \lambda \cdot \Delta$. Then*

$$\max_{\tau \in \mathrm{Im}(f)} \Pr\{f(K'_A) = \tau\} = \begin{cases} 1 & \text{if } \Delta = \mathcal{O}; \\ \leq d_\infty(f^{-1}) \cdot |\Omega|^{-1} & \text{otherwise.} \end{cases}$$

Proof. This follows from Lemma 1 and the observation that for $\Delta \neq \mathcal{O}$ and for any $z \in \mathrm{Im}(f)$, the maximum in Lemma 1 may not be realized for all $K \in f^{-1}(z)$ simultaneously (hence, the inequality for $\Delta \neq \mathcal{O}$). □

Note 3. Lemma 2 reduces to Lemma 1 if one takes f to be the identity function.

2.2 Key Establishment Schemes

In this section, we apply the results of the previous subsection (§2.1) towards scenarios where a party executes a key agreement scheme resulting in a key in group $\mathbb{E}$ and where it also may verify a set of equations in $\mathbb{G}$ (such as those arising from checking ECDSA*certificate verification equations – to be discussed in §2.3). We also consider the scenario where the shared key is mapped from $\mathbb{E}$ towards another group (such as is the case when mapping a computed elliptic curve point to its x-coordinate, which then forms the real shared key (cf., e.g., [2,15])).

Lemma 4. *Consider a key agreement scheme $\mathcal{C}$ between two parties A and B resulting in the establishment of a shared key in group $\mathbb{E}$. Let K_A be the shared key computed by A by executing this key agreement scheme. Let $\Delta_1, \ldots, \Delta_t \in \mathbb{G}$. Let $\Omega_1, \ldots, \Omega_t$ be non-empty subsets of ${\mathbb{Z}_n}^*$, each with cardinality M. Consider the modified key agreement scheme $\mathcal{C}'$ that differs from $\mathcal{C}$ solely in that A generates random secret values $\lambda_i \in_R \Omega_i$ for all $1 \leq i \leq t$ and uses the value $K'_A := K_A + \lambda_1\Delta_1 + \cdots + \lambda_t\Delta_t$ (rather than K_A) as the shared key in protocol $\mathcal{C}$ instead. We assume these random values are kept secret and are not reused. Then the following holds:*

1. *If $\Delta_1 = \cdots = \Delta_t = \mathcal{O}$, then protocols $\mathcal{C}$ and $\mathcal{C}'$ have the same outcome.*
2. *Otherwise, the probability that A and B establish a shared key K is at most $1/M$.*

Proof. This follows directly from Lemma 1. □

Lemma 5. *Consider a key agreement scheme $\mathcal{C}$ between two parties A and B resulting in the establishment of a shared key in group $\mathbb{E}$, which is mapped to a derived key using a fixed function f defined on $\mathbb{E}$. Let K_A be the shared key computed by A by executing this key agreement scheme. Let $\Delta_1, \ldots, \Delta_t \in \mathbb{G}$. Let $\Omega_1, \ldots, \Omega_t$ be non-empty subsets of ${\mathbb{Z}_n}^*$, each with cardinality M. Consider the modified key agreement scheme $\mathcal{C}'$ that differs from $\mathcal{C}$ solely in that A generates random secret values $\lambda_i \in_R \Omega_i$ for all $1 \leq i \leq t$ and uses the value $K'_A := K_A + \lambda_1\Delta_1 + \cdots + \lambda_t\Delta_t$ (rather than K_A) as the shared key in protocol $\mathcal{C}$ instead. We assume these random values are kept secret and are not reused. Then the following holds:*

1. *If $\Delta_1 = \cdots = \Delta_t = \mathcal{O}$, then protocols $\mathcal{C}$ and $\mathcal{C}'$ have the same outcome.*
2. *Otherwise, the probability that A and B establish the same derived key k is at most $d_\infty(f^{-1})/M$.*

Proof. This follows directly from Lemma 4, using Lemma 2. □

2.3 ECDSA and the Modified Signature Scheme ECDSA*

In this section, we briefly summarize those properties of the signature scheme ECDSA [1,9,11] and the modified signature scheme ECDSA*presented in [3] that are relevant to the remainder of the exposition of this paper. For all other details, we refer to the referenced papers themselves.

The ECDSA and ECDSA signature schemes*
With the ECDSA signature scheme, the signature is a pair (r, s) of integers in ${\mathbb{Z}_n}^*$, where the value r is derived from an ephemeral public key R generated by the signer during signature generation via a fixed public function. The ECDSA*signature scheme is the same as ECDSA, except that it outputs the signature (R, s), rather than (r, s).

With the ECDSA*signature scheme, the signature (R, s) over some message m produced by some entity with signature verification key Q may only be valid if $R \in \mathbb{G}$ and if $\Delta := \mathcal{O}$, where

$$\Delta := s^{-1}(eG + rQ) - R, \tag{1}$$

and where e and r are derived from message m and from elliptic curve point R, respectively, via some fixed public functions. For future reference, we mention that the mapping from R to r is such that both R and $-R$ map to the same value. Valid signatures also require the check that $r, s \in {\mathbb{Z}_n}^*$.

One can show that a valid ECDSA*signature (R, s) gives rise to a valid ECDSA signature (r, s) and that, conversely, any valid ECDSA signature (r, s) corresponds to some ECDSA*signature (R, s) with the property that the elliptic curve point $R \in \mathbb{G}$ maps to r via the fixed public function referred to above.

Conversion between ECDSA and ECDSA signatures*
ECDSA has the well-known property that (r, s) is a valid ECDSA signature only if $(r, -s)$ is, i.e., the validity of some ECDSA signature (r, s) does not depend on the 'sign' of signature component s. Similarly, (R, s) is a valid ECDSA*signature only if $(-R, -s)$ is. Both results follow from the observation that the elliptic curve point R and its inverse $-R$ map to the same integer r. Another consequence of this observation is that, while an ECDSA*signature corresponds to a unique ECDSA signature, the converse process generally yields *two* candidate solutions, viz. (R, s) and $(-R, s)$, and uniqueness can only be established by ruling out one of these candidate solutions based on some side information available to the verifier. For a detailed discussion of explicit and implicit side information that realizes this 1-1 mapping, we refer to §4.2 of [3].

3 Accelerated Authenticated Key Agreement

In this section, we introduce a method for securely combining the key authentication step and the key computation step of particular public-key key agreement schemes, rather than carrying these out separately, as is heretofore typically the case. Since the online computational cost of a key agreement protocol - and thereby its running time - are dominated by the computational burden of these two steps, one may anticipate that carrying out these steps in one single computation, rather than carrying these out separately, would result in significant efficiency gains. We show that this is indeed the case. Although the main idea applies more broadly, we are mainly interested in scenarios involving elliptic curves.

In what follows, the exposition is relative to a fixed elliptic curve $I\!E$, with (cyclic) subgroup $\mathbb{G}$ of prime order n generated by some base point G and with identity element $\mathcal{O}$. In particular, we assume that key computations involve operations on this elliptic curve.

We consider a scenario where two parties execute an elliptic curve key agreement scheme that involves computing a shared key $K \in I\!E$ and mapping this to a derived key k. Depending on the details of the scheme in question, this derived key is further processed (e.g., passed through a key derivation function) or used in a subsequent key confirmation step. Examples of such protocols include the elliptic curve variant of the original Diffie-Hellman scheme [8], ECMQV [14], and all elliptic curve key agreement schemes specified by NIST [15] and ANSI [2]. With all these schemes, the mapping f from a shared key $K \in I\!E$ to a derived key k is the projective map, which maps an elliptic curve point that is represented in affine coordinates to its x-coordinate. So, in the notation of §2.1, one has $d_\infty(f^{-1}) = 2$ (since an elliptic curve point and its inverse have the same x-coordinate).

Execution of an authenticated key agreement protocol typically involves checking a public key certificate, to vouch for the binding between the long-term key of the communicating party and its private key (via the corresponding public key). In our exposition, we restrict ourselves to ECDSA*certificates (although our techniques may also be used with, e.g., ephemeral Diffie-Hellman with

signed exponents). Verification of such an ECDSA*certificate involves checking the equation $\Delta = \mathcal{O}$, where the signature verification expression Δ is defined by Equation (1) in §2.3. Notice that $\Delta \in \mathbb{G}$ (since one checks that $R \in \mathbb{G}$).

It now follows from Lemma 4 that one can securely combine the computation of a shared key K_A with verifying a number of ECDSA*signature verification equations $\Delta_1 = \mathcal{O}, \ldots, \Delta_t = \mathcal{O}$ (e.g., those arising from checking a certificate chain) by computing

$$K'_A := K_A + \lambda_1 \Delta_1 + \cdots + \lambda_t \Delta_t,$$

since any non-verifying equation would result in a very small probability that a shared key is indeed established (with proper setting of the size M of the randomness source for integers $\lambda_1, \ldots, \lambda_t$). The same result holds if the shared key is subsequently mapped to a derived key (cf. Lemma 5), where the probability increases by at most a factor $d_\infty(f^{-1}) = 2$, and if the derived key k is subsequently used in a key confirmation step between both communicating parties. (The latter relies on scheme-specific cryptographic details regarding the public functions $g_A : \ k \to g_A(k)$, resp. $g_B : \ k \to g_B(k)$ that implement the key confirmation mechanisms – generally, second-preimage resistance of these functions should suffice to carry forward the properties of Lemma 5.)

3.1 Static-ECDH with ECDSA*Signatures

In this section, we illustrate how combining the key computation step in the elliptic curve authenticated public-key key agreement protocol Static-ECDH with the ECDSA*signature verification step may result in significant overall efficiency improvements.

Static-ECDH is an authenticated elliptic curve version of the basic Diffie-Hellman protocol [8], where two parties A and B exchange certified long-term public keys and verify these upon receipt prior to computing a shared Diffie-Hellman key. We assume that the public keys are certified by a CA via an ECDSA*certificate and that these public keys and the CA's public signature verification key are on the same elliptic curve.

In what follows, we analyze the cost of carrying out the ECDH key computation step and the ECDSA*certificate verification step separately and compare this with the cost of combining both steps in one single computation.

A precise analysis is difficult to give, due to the large number of methods available and implementation-specific trade-offs that favor one method over another. We use multiple-point multiplication and use the Non-Adjacent Form (NAF) and Joint Sparse Form (JSF) representations; for details, see [18], [10, Chapter 3, §3.3.3].

We adopt the following notation: By A and D, we denote the cost of a single point addition and point doubling operation, respectively. By m, we denote the bit-length of finite field elements in $\mathbb{F}_q$, i.e., $m := \lceil \log_2 q \rceil$. We assume the elliptic curve $I\!E := E(\mathbb{F}_q)$ to have a small co-factor h, so that $m \approx \lceil \log_2 n \rceil$.

With static-DH, the key computation for A involves computing the shared key

$$K_A := aQ_B, \tag{2}$$

where Q_B is B's long-term public key extracted from its public key certificate (and checked to be on the curve $I\!E$) and where a is A's private key. Key authentication for A involves verifying B's ECDSA*certificate with signature (R, s) and includes checking that $R \in \mathbb{G}$, evaluating the expression

$$\Delta := s^{-1}(eG + rQ) - R, \tag{3}$$

where e is derived from the to-be-signed certificate information (including B's public key Q_B and B's unique name), where r is derived from signature component R via some fixed public function, and where Q is CA's public key, and checking that $\Delta = \mathcal{O}$ (and that $r, s \in {\mathbb{Z}_n}^*$).

With static-DH, A can combine the key computation (2) and the evaluation of expression (3) and instead compute the expression

$$K'_A := K_A + \lambda\Delta = (aQ_B - \lambda R) + ((\lambda e s^{-1})G + (\lambda r s^{-1})Q), \tag{4}$$

where λ is a suitably chosen random element of ${\mathbb{Z}_n}^*$, after first having checked that $R \in \mathbb{G}$ and $Q_B \in I\!E$.

We now compare the cost of computing expressions (2) and (3) and compare this with the cost of computing expression (4) instead.

When representing scalar a in (2) in NAF format, the expected running time of computing the key K_A is approximately $(m/3)A + mD$ operations. Similarly, when representing the scalars es^{-1} and rs^{-1} in (3) in JSF, the expected running time of evaluating the ECDSA*signature verification equation is approximately $(m/2+2)A+mD$ operations. (Here, 2 point additions account for pre-computing $G \pm Q$, which is necessary for using JSF.) As a result, computing the key K_A and checking the signature verification equation $\Delta = \mathcal{O}$ separately has total running time of approximately $(5m/6 + 2)A + 2 \cdot mD$ operations.

When computing the key K'_A in Equation (4) via multiple-point multiplication and representing both the rightmost two scalars $\lambda \cdot es^{-1}$ and $\lambda \cdot rs^{-1}$ and the leftmost two scalars a and $-\lambda$ in JSF, one obtains a total running time of approximately $2 \cdot (m/2 + 2)A + mD = (m + 4)A + mD$ operations. Thus, combining the key computation step and the signature verification step in one single computation results in a reduction of half of the point doubling operations (at the expense of a small increase of approximately $m/6 + 2$ point addition operations). In fact, one can show that one can do even better and avoid this small increase of point additions altogether, by computing $aQ_B - \lambda R$ differently, by choosing λ suitably, as follows. Recall that λ should be picked at random from a set $\Omega \subset {\mathbb{Z}_n}^*$ of size $|\Omega| \approx \sqrt{n}$. Let I be the set of non-zero coordinates of scalar a, when represented in NAF format. Now, let Ω be the set of all integers in ${\mathbb{Z}_n}^*$ with a representation in NAF format that is zero outside this set I. Since all integers in NAF format are unique and since one can expect the NAF weight of a to be approximately $m/3$ ([10, Theorem 3.29]), one has $|\Omega| = 3^{|I|} \approx 3^{m/3} > \sqrt{n}$. As a result, for $\lambda \in_R \Omega$, one might write

$$aQ_B - \lambda R = (a - \lambda^+ + \lambda^-)Q_B - \lambda^+(R - Q_B) - \lambda^-(R + Q_B), \qquad (5)$$

where $\lambda = \lambda^+ + \lambda^-$ and where λ^+ and λ^- denote those components of λ with the same, respectively different, sign as the corresponding components of a. Since for any coordinate, at most one of the scalars in NAF format on the right-hand side of Equation (5) is nonzero, one can compute this expression using the same number of point additions involved in computing just aQ_B with a in NAF format (after pre-computing $R \pm Q_B$). With this modification, the total running time of computing K'_A becomes $(m/2 + 2)A + (m/3 + 2)A + mD = (5m/6 + 4)A + mD$ operations, i.e., using only half of the point doubling operations required with the separate computations (and just 2 additional point additions).

This result could be interpreted as reducing the incremental running time of ECDSA*signature verification to approximately $(m/2+4)A$ operations, from the approximately $(m/2 + 2)A + mD$ operations that would have been required if ECDSA*signature verification would have been carried out separately using the traditional approach (or roughly $(m/2+4)A+(m/2)D$ operations using the accelerated verification technique of [3]). In other words: the new method effectively removes *all* point doubling operations from the incremental cost of ECDSA*signature verification compared to previously known techniques (in all contexts where this can be combined with key computations).

Rough analysis for P-384 curve

We provide a rough analysis of the relative efficiency of combining the key computation step of the elliptic curve based authenticated key agreement schemes static-DH and ECMQV with the verification of ECDSA*signatures, as proposed in this paper, compared to the traditional method of carrying out both steps separately. Our analysis is based on the elliptic curve curve P-384 defined over a 384-bit prime field, as specified by NIST [9]. All NIST prime curves have co-factor $h = 1$, so by Hasse's theorem, the bit-size of the order of the cyclic group $\mathbb{G}$ is approximately $m = 384$. We consider the cost of public-key operations only (and ignore, e.g., hash function evaluations). Moreover, we ignore the cost of converting the computed shared key to a representation using affine coordinates, as required by most standards (cf., eg., [2,15]).

We assume points to be represented using Jacobian coordinates and that the cost S of a squaring in $\mathbb{Z}_q$ is slightly less than the cost M of a multiplication (we take $S \approx 0.8M$). With Jacobian coordinates, one has $A \approx 8M + 3S$ and $D \approx 4M + 4S$ (see [10, Table 3.3]). Substitution of $S \approx 0.8M$ now yields $A \approx 10.4M$ and $D \approx 7.2M$ and, hence, $D/A \approx 0.69$.

With Static-ECDH, the running time of the key computation is roughly $128A + 384D \approx 393A$ operations. With ECDSA*signature verification, the running time using the traditional method is roughly $194A + 384D \approx 459A$ operations and roughly $196A + 192D \approx 328A$ operations with the accelerated method described in [3]. However, if the key computation step of Static-ECDH and the step of checking the ECDSA*signature verification equation are combined, the incremental running time of signature verification becomes roughly only $196A$ operations. As a result, the new method yields a speed-up of the incremental cost of signature verification of roughly 2.3× compared to the separate method (and roughly 1.7×

P-384 curve	ECDH	ECDSA*(incremental cost)		
		Separately		Combined
ECC operations	key	Ordinary	Fast	with ECDH
– Add	128	194	196	196
– Double	384	384	192	–
– Total (normalized)	393	459	328	196

Fig. 1. Comparison of incremental signature verification cost of ECDSA*, when carried out separately from, resp. combined with, the key computation step of Static-ECDH. Total cost is normalized, using Double/Add cost ratio $D/A = 0.69$.

when compared with the accelerated method of [3]). These results are summarized in Figure 1.

One can perform a similar analysis for the authenticated key agreement scheme ECMQV. With ECMQV, the running time of the key computation is roughly $194A+384D \approx 459A$ operations. If the key computation step of ECMQV and the step of checking the ECDSA*signature verification equation are combined, the incremental running time of signature verification becomes roughly only $196A$ operations, as was also the case with Static-ECDH. As a result, the new method again yields a speed-up of the incremental cost of signature verification of roughly 2.3× compared to the separate method (and roughly 1.7× when compared with the accelerated method of [3]). These results are summarized in Figure 2.

P-384 curve	ECMQV	ECDSA*(incremental cost)		
		Separately		Combined
ECC operations	key	Ordinary	Fast	with ECMQV
– Add	194	194	196	196
– Double	384	384	192	–
– Total (normalized)	459	459	328	196

Fig. 2. Comparison of incremental signature verification cost of ECDSA*, when carried out separately from, resp. combined with, the key computation step of ECMQV. Total cost is normalized, using Double/Add cost ratio $D/A = 0.69$.

A summary of the total normalized cost of various methods is shown in Figure 3.

P-384 curve		Key computation + ECDSA*(total cost)		
Total ECC	Key	ECDSA*separately		ECDSA*
operations	computation	Ordinary	Fast	combined
ECDH	393	852	721	588
ECMQV	459	918	787	655

Fig. 3. Comparison of total normalized combined cost of the key computation step of Static-ECDH and ECMQV and that of ECDSA*signature verification. Normalized cost uses Double/Add cost ratio $D/A = 0.69$.

Experimental results on HP iPAQ 3950 platform
The rough analysis above suggests a significant improvement of the incremental cost of ECDSA*signature verification, when combined with the key computation step of elliptic curve based key agreements schemes over the same curve. This begs the question as to whether the perceived efficiency advantage of using RSA certificates with elliptic curve based key agreement schemes still holds, or whether one might as well use elliptic curve based certificates instead, using our new method. To test this hypothesis, we implemented the various incremental methods of verifying ECDSA*signatures discussed in this paper on an HP iPAQ 3950, Intel PXA250 processor running at a 400MHz clock rate and compared the computational cost hereof with that of RSA signature verification. We compared implementation cost for various cryptographic bit strengths, using the guidance for cryptographic algorithm and key size selection provided by NIST [16]. Our analysis was based on the elliptic curves defined over a prime field for these bit strengths, as specified by NIST [9]. Since these NIST curves all have co-factor $h = 1$, the bit-size of the order of the cyclic group $\mathbb{G}$ and that of the elliptic curve $I\!E$ are the same. The obtained data is shown in Figure 4.

Crypto strength (bits)	Certificate size[1] (bytes)		Ratio ECC/RSA certificates	Verify – incremental cost (ms)			Ratio ECDSA*verify vs. RSA verify
	ECDSA*	RSA		RSA[2]	ECDSA*		
					ordinary[2]	combined[3]	
80	72	256	4x smaller	1.4	4.0	1.7	0.8x faster
112	84	512	6x smaller	5.2	7.7	3.2	1.6x faster
128	96	768	8x smaller	11.0	11.8	4.9	2.2x faster
192	144	1920	13x smaller	65.8	32.9	13.7	4.8x faster
256	198	3840	19x smaller	285.0	73.2	30.5	9.3x faster

Fig. 4. Comparison of (incremental) cost of verifying ECDSA*and RSA certificates, when carried out during the execution of an elliptic curve based authenticated key agreement scheme, for various cryptographic bit strengths and using NIST curves over a prime field. Computational cost considers public-key operations only (and ignores, e.g., hash function evaluation). [1]Excludes non-cryptographic overhead (e.g., identification data) [2]Benchmark Certicom Security Builder™ [3]Estimate.

The implementation benchmarks of Figure 4 suggest a shift of the cross-over point where verification of ECDSA*certificates becomes more efficient than that of RSA certificates, from 128-bit cryptographic bit strength (without the new method) towards roughly 80-bit cryptographic bit-strength (with the new method). Thus, the efficiency advantage one once enjoyed using RSA-based certificates with discrete logarithm based key agreement schemes is no more: one might as well use an 'elliptic curve only' scheme using ECDSA-based certificates instead.

While we did not conduct experiments with ordinary discrete logarithm based key agreement schemes and DSA-based certificates, we speculate one can draw similar conclusions here.

3.2 Extension of Results to ECDSA

The acceleration techniques in this paper depend on signature schemes for which verification can be described as verifying an algebraic equation. An example hereof is ECDSA*, for which verification requires checking Equation (1). The results do not directly apply to ECDSA – a much more widely used signature scheme. To make our efficiency improvements applicable to ECDSA as well, one could map an ECDSA signature (r, s) to an ECDSA*signature (R, s). As explained in §2.3, this mapping is generally a 2-to-1 mapping (since R and $-R$ map to the same integer r). However, there are techniques to eliminate all but one candidate points R that map to the integer r, thus realizing the required 1-1 mapping. Below, we describe various techniques for doing so.

This side information may be appended explicitly to the communicated signature or may be established implicitly, e.g., by generating ECDSA signatures so as to allow unique recovery of the ephemeral point R corresponding to signature component r and used during signing (in particular, it may be realized by changing the sign of signature component s based on inspection of the ephemeral point R used during signing). Since *any* party can change the sign of signature component s of the ECDSA signature without impacting the validity hereof, this procedure can be implemented as post-processing operation by any party and does not necessarily have to be part of the signing process (in particular, this can be done to all legacy ECDSA signatures, to ensure unique conversion hereof into ECDSA*signatures, so as to make the efficiency improvements discussed in this paper and in [3] directly applicable). More specifically, any party who wishes to engage in an authenticated key agreement protocol can bring his public-key certificate into the required format first (as a one-time pre-processing step). While explicitly added side information changes the format of ECDSA, use of implicit side information does not and, therefore, can be used without requiring changes to standardized specifications of ECDSA.

4 Conclusions

We presented a simple technique for combining the key computation step of elliptic curve based key agreement schemes, such as static-ECDH, with the step of verifying particular digital signatures, such as ECDSA*. We demonstrated that the technique easily extends to most standardized elliptic curve based schemes [2,15] using ECDSA, a much more widely used scheme. We demonstrated that this extremely simple technique may result in spectacular computational improvements of overall protocols in which both key computations and, e.g., signature verifications play a role.

We illustrated that the combined key computation and key authentication technique, while exemplified in terms of a single elliptic curve key computation and a single ECDSA signature verification, can be easily generalized towards combined key computations and batch verifications: rather than computing a key K and verifying each of a set of t elliptic curve equations $\Delta_1 = O, \ldots, \Delta_t = O$ separately, simply compute the quantity $K' := K + \lambda_1 \Delta_1 + \cdots \lambda_t \Delta_t$ instead,

where each of $\lambda_1, \ldots, \lambda_t$ is drawn at random from a suitably chosen subset of ${\mathbb{Z}_n}^*$ of size roughly $M \approx \sqrt{n}$. This generalizes the batch verification technique discussed in, e.g., [4,6,7] and covers, e.g., scenarios where one wishes to verify multiple ECDSA signatures (as with certificate chains) or, more generally, any set of elliptic curve equations (batch verification). Moreover, the technique obviously generalizes to settings where one wishes to combine verifications with the computation of a non-secret value (rather than a key), as long as the correctness hereof can be checked.

The approach is not restricted to combining key authentication and key establishment steps during the execution of a key agreement protocol; it can be applied to any setting where one has to compute some keys and verify some elliptic curve equations (provided computations take place on the same curve). This being said, the approach is most useful in settings where one can verify whether the computed key is correct (since this would yield a verdict on whether the homogeneous elliptic curve equation $\Delta = \mathcal{O}$ holds). Thus, the approach works effectively in all settings where one indeed evidences knowledge of the computed key.

As a final note, we would like to emphasize that the presented efficiency improvements do depend on the Double/Add cost ratio D/A of the underlying curve in question. Our analysis in §3.1 used NIST curves defined over a prime field and Jacobian coordinates. Careful analysis is needed before one were to apply results directly to, e.g., binary non-Koblitz curves, curves using different coordinate systems, or, e.g, (twisted) Edwards curves [5]. As an example, for twisted curves, Table 1 of [13] suggests one has $A \approx 7M$ and $D \approx 4M + 3S$. Substitution of $S \approx 0.8M$ now yields $A \approx 7M$ and $D \approx 6.4M$ and, hence, $D/A \approx 0.91$. If so, the new method of this paper become more pronounced and would yield a $2.85\times$ efficiency improvement of the incremental ECDSA* verification cost compared to the traditional ECDSA signature verification approach, whereas the acceleration technique in [3] yields a $1.75\times$ efficiency improvement hereto. Since, to our knowledge, this is still very much a nascent research area, this seems to be a promising area of future research.

References

1. ANSI X9.62-1998. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standard for Financial Services. American Bankers Association (January 7, 1999)
2. ANSI X9.63-2001. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography. American National Standard for Financial Services. American Bankers Association (November 20, 2001)
3. Antipa, A., Brown, D.R., Gallant, R., Lambert, R., Struik, R., Vanstone, S.A.: Accelerated Verification of ECDSA Signatures. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 307–318. Springer, Heidelberg (2006)
4. Bellare, M., Garay, J.A., Rabin, T.: Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)

5. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Twisted Edwards Curves. International Association for Cryptologic Research, IACR e/Print 2008-013
6. Cao, T., Lin, D., Xue, R.: Security Analysis of Some Batch Verifying Signatures from Pairings. International Journal of Network Security 3(2), 138–143 (2006)
7. Cheon, J.H., Lee, D.H.: Use of Sparse and/or Complex Exponents in Batch Verification of Exponentiations. International Association for Cryptologic Research, ePrint 2005/276 (2005)
8. Diffie, W., Hellmann, M.E.: New Directions in Cryptography. IEEE. Trans. Inform. Theory IT-22, 644–654 (1976)
9. FIPS Pub 186-3. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-3. US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, Maryland, USA (February 2009), Includes change notice (October 5, 2001)
10. Hankerson, D.R., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. Springer, New York (2003)
11. Johnson, D.J., Menezes, A.J., Vanstone, S.A.: The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security 1, 36–63 (2001)
12. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger Security for Authenticated Key Exchange. International Association for Cryptologic Research, ePrint 2006/073 (2006)
13. Longa, P., Gebotys, C.: Efficient Techniques for High-Speed Elliptic Curve Cryptography. International Association for Cryptologic Research, IACR e/Print 2010-315
14. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An Efficient Protocol for Authenticated Key Agreement. Centre for Applied Cryptographic Research, Corr 1998-05, University of Waterloo, Ontario, Canada (1998)
15. NIST Pub 800-56a. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised). NIST Special Publication 800-56A. US Department of Commerce/National Institute of Standards and Technology, Springfield, Virginia (March 8, 2007)
16. NIST Pub 800-57. Recommendation for Key Management – Part 1: General (Revised), NIST Special Publication 800-57. US Department of Commerce/National Institute of Standards and Technology, Springfield, Virginia (March 8, 2007)
17. Proos, J.: Joint Sparse Forms and Generating Zero Columns when Combing. Centre for Applied Cryptographic Research, Corr 2003-23, University of Waterloo, Ontario, Canada (2003)
18. Solinas, J.: Low-Weight Binary Representations for Pairs of Integers. Centre for Applied Cryptographic Research, Corr 2001-41, University of Waterloo, Ontario, Canada (2001)