2014 IEEE 27th Computer Security Foundations Symposium

Certified Synthesis of Efficient Batch Verifiers

Joseph A. Akinyele*, Gilles Barthe†, Benjamin Grégoire‡, Benedikt Schmidt†, and Pierre-Yves Strub†

* Johns Hopkins University & Zeutro LLC, Baltimore, USA
†IMDEA Software Institute, Madrid, Spain
‡INRIA Sophia-Antipolis Méditerranée, France

***Abstract*—Many algorithms admit very efficient batch versions that compute simultaneously the output of the algorithms on a set of inputs. Batch algorithms are widely used in cryptography, especially in the setting of pairing-based computations, where they deliver significant speed-ups.**

AutoBatch is an automated tool that computes highly optimized batch verification algorithms for pairing-based signature schemes. Thanks to finely tuned heuristics, AutoBatch is able to rediscover efficient batch verifiers for several signature schemes of interest, and in some cases to output batch verifiers that outperform the best known verifiers from the literature. However, AutoBatch only provides weak guarantees (in the form of a LaTeX proof) of the correctness of the batch algorithms it outputs. In this paper, we verify the correctness and security of these algorithms using the EasyCrypt framework. To achieve this goal, we define a domain-specific language to describe verification algorithms based on pairings and provide an efficient algorithm for checking (approximate) observational equivalence between expressions of this language. By translating the output of AutoBatch to this language and applying our verification procedure, we obtain machine-checked correctness proofs of the batch verifiers. Moreover, we formalize notions of security for batch verifiers and we provide a generic proof in EasyCrypt that batch verifiers satisfy a security property called screening, provided they are correct and the original signature is unforgeable against chosen-message attacks. We apply our techniques to several well-known pairing-based signature schemes from the literature, and to Groth-Sahai zero-knowledge proofs.

I. Introduction

Designing efficient and provably secure cryptographic constructions is a central goal in cryptography. While proposing new constructions is the prevailing route to achieving this goal, cryptography also has a long history of optimization techniques that improve the efficiency of existing constructions. Batching is one such technique; informally, a batch version of an algorithm is another algorithm which runs multiple instances of the original (non-batch) algorithm on different inputs. While all algorithms admit a batch version, simply by iterating the algorithm on all elements of the input set, batch algorithms can be significantly more efficient than the naive ones. The efficiency gained by relying on batch algorithms is particularly critical in scenarios where algorithms must be executed efficiently under stringent computational resource constraints. See for instance [59] for a successful application of batch verification to vehicular sensor networks.

There is a long tradition of developing batch verifiers in cryptography. One influential work is the one of Bellare, Garay and Rabin [11], who propose three generic techniques for building efficient batch verifiers: the random subset test, the small exponents test, and the bucket test. Later, Cha and Cheon propose [27] two additional techniques: the sparse exponent test, and the complex exponent test, based on similar ideas. When formulated in the setting of an additive group, these techniques exploit the idea of using randomization to reduce verification of an arbitrary number of equations to verification of a single equation on random linear combinations of the equations. The techniques induce correct batch verifiers in the sense that: i. whenever the original algorithm accepts, the batch algorithm accepts; ii. whenever the original algorithm rejects at least one input, the batch verification algorithm rejects *except with negligible probability*.

Pairing-based cryptography is the branch of cryptography that studies cryptographic constructions based on bilinear pairings; over the last fifteen years, pairing-based cryptography has received considerable attention. On the one hand, there has been significant effort to find efficient pairings based on elliptic or hyperelliptic curves. On the other hand, pairings have been used as a building block in numerous cryptographic constructions. Early instances of such constructions include the three-party one-round key exchange protocol by Joux [45], the identity-based encryption scheme by Boneh and Franklin [17], or the short signatures by Boneh, Lynn, and Shacham [18]. More recent constructions based on pairings include the efficient non-interactive zero-knowledge (NIZK) proofs by Groth and Sahai [38] or Jutla and Roy [46], as well as the structure-preserving signatures by Abe, Fuchsbauer, Groth, Haralambiev and Ohkubo [2].

Pairing-based computations are a particularly interesting application of batch verification because computing with pairings is very expensive and thus verification of pairing-based computations can be inefficient. Consequently, there has been a significant amount of work to adapt generic batching techniques to the bilinear setting and to develop batch verifiers for existing verification algorithms for pairing-based signature schemes [18], [25], [58], [60], [61], [21], [33] or NIZK proofs [35], [14], [24]. This line of work has resulted in very efficient batch verifiers requiring a constant number of pairings that is independent of the number of checked signatures. However, it remains a challenge to find optimal batch verifiers for pairing-based computations. Indeed, efficient batch verifiers are typically derived by combining the aforementioned generic techniques with algebraic reasoning about the underlying mathematical objects, in this case pairing-based computations. Since algebraic reasoning on such computations is quite complex, deriving a correct batch verifier is error-prone; moreover, due to the large search space, manual construction of batch verifiers might lead to sub-optimal algorithms.

DOI 10.1109/CSF.2014.19

AutoBatch [5] is an automated tool for generating efficient batch verifiers from high-level descriptions of pairing-based signature schemes. In particular, the tool takes as input a Scheme Description Language (SDL) description of a signature scheme and applies a series of techniques to produce optimized batch verifiers. Additionally, the tool generates working implementations of the resulting batch verifiers in either Python or C++ using Charm [3].

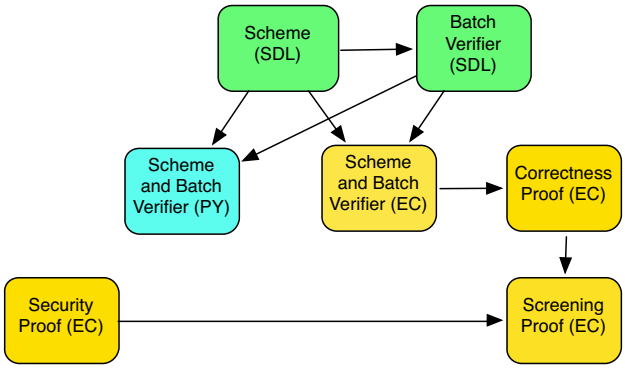
AutoBatch demonstrates by an extensive analysis of algorithms from the literature that it is feasible to synthesize automatically optimized batch verifiers, and that in some cases the output verifiers are more efficient than manually crafted verifiers. More generally, AutoBatch supports the view that the design and optimization of cryptographic constructions can greatly benefit from computer-aided tools. However, AutoBatch addresses only partially the key concern of delivering provably secure constructions; indeed, AutoBatch only generates a LaTeX proof of correctness for the batch verifier, leaving the generation of a machine-checkable proof as an open problem for further work.

In this article, we address this issue and provide support for certifying the correctness and security of batch verifiers output by AutoBatch. Concretely, we develop a certifying back-end for AutoBatch; the back-end is based on EasyCrypt [10], a framework for the verification of probabilistic programs. EasyCrypt has been used extensively to verify the security of cryptographic constructions. In this paper, we show that EasyCrypt is sufficiently versatile to reason about the correctness of the optimizations performed by AutoBatch, as well as about the security of the batch verifiers synthesized by AutoBatch. The EasyCrypt back-end consists of two main components: the first component takes as input a verification algorithm and a batch verifier, and certifies the correctness of the latter. This component is based on an automated procedure for checking equality of expressions built from bilinear maps and group operations. The other component is a generic proof that a correct batch verifier satisfies a notion of security called screening (originally introduced in [11]), provided the original verifier is secure against chosen-message attacks. The resulting workflow is described in Figure 1.

Moreover, we initiate the extension of our approach to zero-knowledge proofs for pairing-based statements. In [38], Groth and Sahai develop an approach to build efficient non-interactive zero-knowledge (NIZK) and witness-indistinguishable (NIWI) proofs of satisfiability for several important classes of pairing-based equations. Since their inception, Groth-Sahai proofs have been widely adopted in the context of pairing-based cryptography, allowing the realization of many new constructions, including group and ring signatures, voting, anonymous broadcast, anonymous credentials and verifiable encryption. Later, Blazy et al. [14] show how batch verification can lead to significant speedups, especially for instantiations based on the symmetric external Diffie-Hellman SXDH assumption and the decisional linear DLIN assumption. We present an extension of our framework that is able to automatically compute and certify batch Groth-Sahai proofs of this type.

Contributions

To achieve this goal, we make the following contributions:



AutoBatch starts with a full description of a signature scheme in SDL. AutoBatch then searches for an optimized batch verifier and produces it in SDL and outputs Python (and/or C++) implementations for both algorithms. The SDL representations of the scheme and batch verifier are sent to EasyCrypt, and formally verified automatically. The user can optionally provide an EasyCrypt proof of unforgeability of the scheme, from which screening security of the batch verifier is derived automatically.

Fig. 1. Overall architecture

1) We define a domain-specific language for describing verification algorithms built from computations in bilinear groups, equality checks, random sampling, and bounded loops (modelled using big operators). Our language is sufficiently expressive to capture most such algorithms (both batch and non-batch) from the literature.
2) We develop a procedure to reason about equality of deterministic expressions of our domain-specific language. Moreover, we show that the techniques of [5], which formalize most existing optimization techniques from the cryptographic literature, can be validated automatically by our verification procedures. Using eager sampling (which transforms an arbitrary expression into an equivalent expression that performs all its samplings upfront and then makes a deterministic computation), we extend our procedure to arbitrary expressions.
3) We formalize the Small Exponent Test for prime order groups. This test is key to combining multiple equations into a single one, by first randomizing each equation and then combining them. We also prove the correctness of the Small Exponent Test for composite order groups whose order has no small prime factor.
4) We formalize a meta-theorem in EasyCrypt that relates the so-called screening security of a signature scheme with the security of a batch scheme obtained using our transformations. More precisely, we show that if the original scheme is secure with respect to unforgeability for chosen message attacks (UF-CMA), then any correct batch verifier for that scheme is secure against screening attacks.
5) We combine AutoBatch with EasyCrypt to obtain certified optimized batch algorithms for signature schemes. We demonstrate the usefulness of our toolchain by certifying the examples from AutoBatch. Moreover, we prove UF-CMA security of the well-known CL signature scheme from [22] and obtain automatically from the proof

of correctness generated by our tool and the proof of the meta-theorem a proof of screening security for batch-CL.
6) We extend AutoBatch to output efficient batch verifiers for Groth-Sahai proofs, and we use EasyCrypt to automatically verify the correctness of the batch verifiers.

II. Notation

Following [31], and for the clarity of exposition, we use additive notation for the groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Note however that the literature on pairing-based cryptography, including the paper on AutoBatch [5], often uses multiplicative notation for the source groups. Moreover, we use g_1, g_2 and g_T to denote the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively, and $[u]a$ for scalar multiplication. We say a pairing is symmetric if there exists an efficient isomorphism between the two source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ (abstractly, $\mathbb{G}_1 = \mathbb{G}_2$), and asymmetric otherwise. In the remainder of the paper, all propositions have been proved in EasyCrypt unless stated otherwise.

III. Motivating example: batching CL signatures

A. Public-key signature schemes

A signature scheme consists of four algorithms $(\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$. The setup algorithm ParamGen is a probabilistic algorithm that takes as input the security parameter and outputs the mathematical structures for running the algorithm. Key generation KGen is another probabilistic algorithm that takes as input the setup and computes the secret and public keys of a user. Signing Sign is a probabilistic algorithm that takes as input a message and a secret key and returns a signed message. Finally, verification Ver is a (often deterministic) algorithm that takes as input a public key pk, a message m, and a signed message ms and checks whether ms is a valid signature of m for the public key pk. A signature scheme is required to comply with the following correctness condition, which ensures that signatures generated correctly are always accepted by the verifier, i.e,

$$\Pr\left[\left(\begin{array}{l}\mathbf{P} \leftarrow \mathsf{ParamGen}(1^\eta);\\ (sk, pk) \leftarrow \mathsf{KGen}(\mathbf{P});\\ \sigma \leftarrow \mathsf{Sign}(sk, m);\\ b \leftarrow \mathsf{Ver}(pk, m, \sigma)\end{array}\right) : b = \mathrm{ok}\right] = 1$$

The CL signature scheme is a pairing-based signature scheme introduced by Camenisch and Lysyanskaya [22]. It is defined as follows:

- ParamGen returns a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T, e)$, consisting of a prime p, cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of order p, generators g_1, g_2 and g_T and a polynomial-time bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that $\mathsf{e}(g_1, g_2) = g_T$ and for all $a \in \mathbb{G}_1$, $b \in \mathbb{G}_2$ and $u, v \in \mathbb{Z}$, $\mathsf{e}([u]a, [v]b) = [u \cdot v]\ \mathsf{e}(a, b)$;
- KGen samples x and y uniformly in $\mathbb{Z}_p$ and returns as secret key $sk = (x, y)$ and as public key $pk = (X, Y)$ where and $X = [x]g_1$ and $Y = [y]g_1$;
- Sign takes as input the secret key sk and a message $m \in \mathbb{Z}_p^*$, samples r uniformly in $\mathbb{Z}_p^*$ and returns the signature $([r]g_2, [ry]g_2, [r(x + mxy)]g_2) \in \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_2$;
- Ver takes as input the public key pk, a message $m \in \mathbb{Z}_p^*$ and a signed message $(a, b, c) \in (\mathbb{G}_2 \setminus \{0\}) \times \mathbb{G}_2 \times \mathbb{G}_2$ and verifies the following equations:

$$\mathsf{e}(Y, a) = \mathsf{e}(g_1, b) \qquad \mathsf{e}(X, a) + [m]\mathsf{e}(X, b) = \mathsf{e}(g_1, c)$$

B. Unforgeability against chosen-message attacks

The standard notion of security for public-key signature schemes is existential unforgeability against chosen-message attacks (UF-CMA), which ensures an adversary has small probability of forging a valid signature for a message of its choice. Formally, the advantage $\mathsf{Adv}_{\mathsf{UF\text{-}CMA}(\mathbf{S})}(\mathcal{A})$ of an adversary against existential unforgeability of chosen-message attacks for a signature scheme $\mathbf{S} = (\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ is defined as

$$\Pr\left[\left(\begin{array}{l}\mathbf{P} \leftarrow \mathsf{ParamGen}(1^\eta);\\ (sk, pk) \leftarrow \mathsf{KGen}(\mathbf{P});\\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}}(pk);\\ b \leftarrow \mathsf{Ver}(pk, m, \sigma)\end{array}\right) : b = \mathrm{ok} \wedge m \notin \boldsymbol{L}\right]$$

where $\boldsymbol{L}$ denotes the list of signature queries performed by the adversary $\mathcal{A}$. Moreover, the signature scheme $\mathbf{S}$ is (t, q, ϵ)-secure against existential unforgeability of chosen-message attacks, or (t, q, ϵ) UF-CMA-secure for short, if for adversary $\mathcal{A}$ executing in time at most t and performing at most q requests to the signing oracle,

$$\mathsf{Adv}_{\mathsf{UF\text{-}CMA}(\mathbf{S})}(\mathcal{A}) \leq \epsilon$$

The CL signature scheme can be proved UF-CMA secure under the LRSW assumption [50]. The advantage $\mathsf{Adv}_{\mathsf{LRSW}}(\mathcal{B})$ of an adversary $\mathcal{B}$ against the LRSW assumption is defined as

$$\Pr\left[\left(\begin{array}{l}\mathbf{P} \leftarrow \mathsf{ParamGen}(1^\eta);\\ (sk, pk) \leftarrow \mathsf{KGen}(\mathbf{P});\\ (m, a, b, c) \leftarrow \mathcal{B}^{\mathcal{O}}(pk);\end{array}\right) : \begin{array}{l}\mathbf{Eqs} \wedge m \notin \boldsymbol{L} \wedge\\ a \neq 0 \wedge m \neq 0\end{array}\right]$$

where **Eqs** is defined as $b = [y]a \wedge c = [x + mxy]a$, $\boldsymbol{L}$ is the list of queries to the oracle $\mathcal{O}$, and the oracle $\mathcal{O}$ takes as input $m \in \mathbb{Z}_q$, samples r uniformly in $\mathbb{G}_2$ and returns $(r, [y]r, [x + mxy]r) \in \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_2$.

Proposition III.1 (Unforgeability of CL signatures) For every adversary $\mathcal{A}$ against UFCMA-security of CL signatures that executes in time t and makes at most q queries to the signing oracle, one can construct an adversary $\mathcal{B}$ against the LRSW assumption that executes in time $O(t)$ and makes at most q queries to the oracle $\mathcal{O}$, such that

$$\mathsf{Adv}_{\mathsf{UF\text{-}CMA}(CL)}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{LRSW}}(\mathcal{B})$$

Sketch: The reduction is straightforward: the adversary $\mathcal{B}$ forwards signatures queries to the oracle $\mathcal{O}$ and returns the output of $\mathcal{A}$. We must show that if the output of $\mathcal{A}$ passes verification, then it satisfies **Eqs**, which can be done directly by algebraic calculations. ■

Hence UF-CMA security of CL signatures follows from the hardness of the LRSW assumption.

C. Batch verification

A batch verifier for a signature scheme is an algorithm that verifies simultaneously multiple signatures, and satisfies the following two properties: if all signatures in a submitted batch are valid, then the batch verifier accepts, and; if some signature in the submitted batch is invalid, then it rejects with overwhelming probability. Formally, an algorithm BVer is an ϵ-batch verifier for a signature scheme $(\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ iff for every $\mathbf{w} = \overrightarrow{(pk, m, \sigma)}$:

$$0 \leq \Pr[b \leftarrow \mathsf{BVer}(\mathbf{w}) : b = \mathrm{ok}] - \Pr[b \leftarrow \mathsf{Ver}^b(\mathbf{w}) : b = \mathrm{ok}] \leq \epsilon$$

where Ver^b iteratively applies the verification algorithm to all triples $\mathbf{w}[i]$. This definition is inspired from [21]; however, our definition also considers the case where the verification algorithm is probabilistic. In this case, our definition ensures that the rejection probability of BVer is upper-bounded by the rejection probability of Ver^b, i.e., the batch verifier does not reject more often than the iterated verification algorithm. Moreover, note that our definition considers the case of batch verification for multiple signers; the case of single signers is similar, except that $\mathbf{w}$ is now of the form $(pk, \overrightarrow{(m, \sigma)})$. Both definitions are instances of a more general notion of batch verifier that applies to arbitrary algorithms returning a boolean value; see Section IV.

A simple-minded approach to obtain efficient batch verifiers is to combine all verification equations into a single one using the group laws. Following this approach, a batch verifier for a set of message and signature pairs $(m_i, (a_i, b_i, c_i))_{1 \leq i \leq n}$ from a single user with public key (X, Y) would simply check:

$$\sum_{1 \leq i \leq n} \mathsf{e}(Y, a_i) = \sum_{1 \leq i \leq n} \mathsf{e}(g_1, b_i)$$
$$\sum_{1 \leq i \leq n} (\mathsf{e}(X, a_i) + [m_i]\mathsf{e}(X, b_i)) = \sum_{1 \leq i \leq n} \mathsf{e}(g_1, c_i))$$

However, such a verifier is incorrect; indeed, it is immediate for an adversary to produce a batch that is accepted by this algorithm but contains invalid signatures. This can be avoided by sampling values δ_i, δ'_i uniformly over $(0 \ldots 2^\ell - 1)$ and by verifying:

$$\sum_{1 \leq i \leq n} [\delta_i]\mathsf{e}(Y, a_i) + [\delta'_i](\mathsf{e}(X, a_i) + [m_i]\mathsf{e}(X, b_i)) = \sum_{1 \leq i \leq n} [\delta_i]\mathsf{e}(g_1, b_i) + [\delta'_i]\mathsf{e}(g_1, c_i)$$

This so-called Small Exponent Test has a small probability to accept a bad signature, precisely $2^{-\ell}$; thus, the choice of ℓ is a trade-off between security and efficiency; a reasonable choice for ℓ is 80, so that the error is 2^{-80}. The correctness of the Small Exponent Test is captured by the following proposition.

Proposition III.2 (Small Exponent Test) Let p be a prime and let $a_i, b_i \in \mathbb{Z}_p$ for $i < n$. Consider the expressions $s = \sum_{i<n}[x_i]a_i$ and $t = \sum_{i<n}[x_i]b_i$, where x_i are chosen uniformly at random over $(0 \ldots 2^{\ell-1})$, where $2^\ell \leq p$. Then the probability that $s = t$ with $a_i \neq b_i$ for some i is upper bounded by $2^{-\ell}$.

For the purpose of CL signatures, we only use the test for prime order groups. As noted by [11], primality is important for the validity of the test; in general, the test fails for arbitrary groups. However, we show in Section IV-C that the Small Exponent Test remains valid for an important class of composite order groups.

The next step to obtain an efficient batch verifier is to perform equational reasoning in order to transform the above equations into equations with a smaller number of calls to the bilinear map—these computations dominate the runtime, and thus the main purpose of the optimization phases is to minimize their usage. AutoBatch identifies a series of transformations and implements heuristics that lead to more efficient algorithms. Figure 2 shows an excerpt of the sequence of transformations performed by AutoBatch to obtain an efficient batch verifier.

The next step is to compile into EasyCrypt the signature scheme and the batch verifier output by AutoBatch. The EasyCrypt file output by the translation contains various typing declarations and axioms that capture the expected properties of the parameters. The translation is based on a library for bilinear maps. Then, the correctness of the batch verifier is proved automatically as follows. First, we build the naive batch verifier from the original equation, and generate a proof of correctness in EasyCrypt; the latter is obtained by instantiating a generic proof, taking advantage of the (newly developed) module system of EasyCrypt. Then, we apply the Small Exponent Test to the original equation; the Small Exponent Test is proved generically in EasyCrypt, and thus this step is justified simply by instantiation. Then, one invokes an automated procedure that proves the equivalence between the expression obtained by applying the Small Exponent Lemma, and the batch verifier output by AutoBatch. Pleasingly, the automated procedure is able to automatically prove the equivalence, without additional help.

D. Screening property of batch verifiers

Bellare, Garay and Rabin [11] introduce an alternative notion to batch verification, called screening, which can be achieved more efficiently, and is sometimes sufficient in practice. The advantage $\mathsf{Adv}_{\mathsf{Screen}(\mathbf{S},\mathsf{BVer})}$ of an adversary $\mathcal{A}$ against screening security of a batch verifier BVer for a signature scheme $\mathbf{S} = (\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ is defined as

$$\Pr\left[\begin{pmatrix} p \leftarrow \mathsf{ParamGen}(1^\eta); \\ (sk_0, pk_0) \leftarrow \mathsf{KGen}(p); \\ (i, \overrightarrow{pk, m, \sigma}) \leftarrow \mathcal{A}^{\mathsf{Sign}}(pk_0); \\ b \leftarrow \mathsf{BVer}(\overrightarrow{pk, m, \sigma}); \end{pmatrix} : b = \mathrm{ok} \wedge \mathbf{Forg}(i)\right]$$

where $\mathbf{Forg}(i)$ denotes $\vec{pk}[i] = pk_0 \wedge \vec{m}[i] \notin \boldsymbol{L}$ and $\boldsymbol{L}$ is the list of signature queries performed by the adversary $\mathcal{A}$.

One can relate unforgeability of the original signature scheme and screening security of the batch verifier scheme as follows.

Proposition III.3 Let $(\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme that is (t, q, ϵ)-unforgeable against chosen-message attacks. If the algorithm BVer is a ϵ'-batch verifier for $(\mathsf{ParamGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$, then the advantage

We begin with the original verification equation.

$\mathsf{e}(Y,a) \doteq \mathsf{e}(g,b)$ and $\mathsf{e}(X,a) + [m]\mathsf{e}(X,b) \doteq \mathsf{e}(g,c)$

Step 1: Consolidate the verification equations (tech. 0a), and apply the small exponents test as follows: For each of the $i = 1$ to η signatures, choose random $\delta_1, \delta_2 \in [1, 2^\lambda - 1]$ and compute for each equation:

$[\delta_1]\mathsf{e}(g,b) + [-\delta_1]\mathsf{e}(Y,a) \doteq [\delta_2]\mathsf{e}(X,a) + [m \cdot \delta_2]\mathsf{e}(X,b) + [-\delta_2]\mathsf{e}(g,c)$

Step 2: Combine η signatures (tech. 1), move the scalar(s) inside pairing (tech. 2):

$$\sum_{i=1}^{\eta} \mathsf{e}(g,[\delta_{i,1}]b_i) + \mathsf{e}(Y,[-\delta_{i,1}]a_i) \doteq \sum_{i=1}^{\eta} \mathsf{e}(X,[\delta_{i,2}]a_i) + \mathsf{e}(X,[m_i \cdot \delta_{i,2}]b_i) + \mathsf{e}(g,[-\delta_{i,2}]c_i)$$

Step 3: Merge pairings with common first or second argument (tech. 3b):

$$\sum_{i=1}^{\eta} \mathsf{e}(g,[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \mathsf{e}(Y,[-\delta_{i,1}]a_i) \doteq \sum_{i=1}^{\eta} \mathsf{e}(X,[\delta_{i,2}]a_i) + \mathsf{e}(X,[m_i \cdot \delta_{i,2}]b_i)$$

Step 4: Merge pairings with common first or second argument (tech. 3b):

$$\sum_{i=1}^{\eta} \mathsf{e}(g,[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \mathsf{e}(Y,[-\delta_{i,1}]a_i) \doteq \sum_{i=1}^{\eta} \mathsf{e}(X,[\delta_{i,2}]a_i + [m_i \cdot \delta_{i,2}]b_i)$$

Step 5: Move sum inside pairings to reduce η pairings to 1 (tech. 3a):

$$\sum_{i=1}^{\eta} \mathsf{e}(g,[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \mathsf{e}(Y,[-\delta_{i,1}]a_i) \doteq \mathsf{e}(X,\sum_{i=1}^{\eta}[\delta_{i,2}]a_i + [m_i \cdot \delta_{i,2}]b_i)$$

Step 6: Distribute sum (tech. 5):

$$\sum_{i=1}^{\eta} \mathsf{e}(g,[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \sum_{i=1}^{\eta} \mathsf{e}(Y,[-\delta_{i,1}]a_i) \doteq \mathsf{e}(X,\sum_{i=1}^{\eta}[\delta_{i,2}]a_i + [m_i \cdot \delta_{i,2}]b_i)$$

Step 7: Move sum inside pairings to reduce η pairings to 1 (tech. 3a):

$$\mathsf{e}(g,\sum_{i=1}^{\eta}[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \mathsf{e}(Y,\sum_{i=1}^{\eta}[-\delta_{i,1}]a_i) \doteq \mathsf{e}(X,\sum_{i=1}^{\eta}[\delta_{i,2}]a_i + [m_i \cdot \delta_{i,2}]b_i)$$

Fig. 2. Selected steps of AutoBatch trace for CL-signatures

$$\begin{array}{rcll}
e & = & z & \text{variable} \\
& | & 1 & \text{one} \\
& | & 0_k & \text{neutral element} \\
& | & g_k & \text{group generator} \\
& | & -e & \text{group inverse} \\
& | & e + e & \text{group law} \\
& | & e \cdot e & \text{multiplication in } \mathbb{Z}_n \\
& | & [e]e & \text{scalar multiplication} \\
& | & \mathsf{e}(e,e) & \text{pairing} \\
& | & e \doteq e & \text{equality test} \\
& | & e \wedge e & \text{conjunction} \\
& | & \sum_{i<m} e & \text{big operator group law} \\
& | & \wedge_{i<m} e & \text{big operator conjunction} \\
& | & e_i & \text{projection} \\
& | & \mathsf{let}\ r \xleftarrow{\$} U\ \mathsf{in}\ e & \text{random sampling} \\
& | & \mathsf{let}\ x \leftarrow e\ \mathsf{in}\ e & \text{let binding}
\end{array}$$

where $k \in \{1, 2, T\}$, m ranges over size variables, i ranges over indexes, z ranges over variables, and U is drawn from the syntax:

$$\begin{array}{rcll}
U & = & \mathbb{Z}_n & \text{integers modulo } n \\
& | & \mathbb{Z}_n^{<\ell} & \text{integers modulo } n \text{ of size } < \ell \\
& | & \mathbb{G}_1 & \text{first source group} \\
& | & \mathbb{G}_2 & \text{second source group} \\
& | & \mathbb{G}_T & \text{target group} \\
& | & U^m & \text{product type}
\end{array}$$

Fig. 3. Expression language for pairing-based computations

$\mathsf{Adv}_{\mathsf{Screen}(\mathsf{S},\mathsf{BVer})}$ of an adversary $\mathcal{A}$ executing in time t and making at most q requests to Sign is upper bounded by $\epsilon + \epsilon'$.

By instantiating the proposition to CL signatures and applying the (interactively developed) proof of unforgeability and the (automatically generated) proof of correctness described above, one obtains an EasyCrypt proof of screening security for CL signatures.

IV. Equivalence of pairing-based computations

This section introduces a domain-specific language for pairing-based computations, provides transformations that preserve the semantics of expressions, except for some small probability, and finally provides a procedure for the approximate equivalence of two expressions. Our results in the this section hold for prime-order and composite-order groups, but our EasyCrypt development is restricted to the prime-order case.

A. Expression language

Expressions of our language are built from group operations, bilinear pairings, boolean operators, comparison, uniform sampling over finite sets, and let definitions. The syntax also features big operators for the group law and boolean conjunction. Formally, the set of expressions is defined by the grammar of Fig. 3. As usual, the big operators and the let operators are binding; we let $\mathrm{FV}(e)$ denote the set of free variables of an expression e. Moreover, we say that an expression e is deterministic, written $\mathsf{det}(e)$, if e does not contain any random sampling.

As an illustration of the syntax, the naive batch verifier for CL signatures with input $((X_i, Y_i), m_i, (a_i, b_i, c_i))_{i<n}$ is defined as

$$\bigwedge_{i<n} (\mathsf{e}(Y_i, a_i) \doteq \mathsf{e}(g_1, b_i) \wedge \mathsf{e}(X_i, a_i) + [m_i]\mathsf{e}(X_i, b_i) \doteq \mathsf{e}(g_1, c_i))$$

We equip the language with a simple type system that isolates meaningful expressions; in addition to the types given in Fig. 3, we also consider the type $\mathbb{B}$ of booleans. Typing judgments are of the form

$$i_1 < m_1, i_2 < m_2 \ldots \mid x : U, x' : U' \ldots \vdash e : V$$

where $i_1, i_2, \ldots$ are indices, $m_1, m_2, \ldots$ are size variables, $x, x' \ldots$ are variables, e is an expression, and $U, U', \ldots, V$ are types. Typing rules are standard, for instance:

$$\frac{i_1 < m_1, \ldots, i_k < m_k \mid x : U, x' : U' \ldots \vdash e : V^{m_s}}{i_1 < m_1, \ldots, i_k < m_k \mid x : U, x' : U' \vdash e_{i_s} : V}$$

assuming that $s \in \{1 \ldots k\}$. Operators are overloaded in the expected way; for instance, $+$ denotes addition in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and $\mathbb{Z}_n$. In the sequel, we use a, b, c for group elements and u, v, w for elements of $\mathbb{Z}_n$ and ϕ, ψ for booleans.

Well-typed expressions are given a probabilistic interpretation as follows. In this paper, we only consider distributions over finite sets, so it is sufficient to view a distribution over X as a function $\mu : X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. In the sequel, we let **let** μ **in** μ' denote the standard bind operator for probability distributions, i.e.

$$\textbf{let } \mu \textbf{ in } M = \lambda y. \sum_{x \in X} \mu(x)\ M(x)(y)$$

The interpretation is defined by the following steps; we consider both the case of prime order groups and the case of composite order groups. First, one interprets size variables and n and ℓ as natural numbers. We require that n is interpreted as a prime number or as the product of large primes, i.e. of the form $p_1 \cdots p_k$ where the p_i are large primes. We also require that 2^ℓ is smaller than n in the first case and smaller than all p_i in the second case. Then, we define a set-theoretical interpretation for types (we let $[\![U]\!]$ denote the interpretation of U) such that $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of order n, and there exists a bilinear pairing $[\![\mathsf{e}]\!] : [\![\mathbb{G}_1]\!] \times [\![\mathbb{G}_2]\!] \to [\![\mathbb{G}_T]\!]$. Finally, we interpret expressions of type U as distributions over $[\![U]\!]$. The interpretation is defined relative to a valuation ρ that maps variables of type U to elements of $[\![U]\!]$. The interpretation is standard, for instance:

- $[\![e_1 + e_2]\!]_\rho$ is defined by the clause
$$\textbf{let } x \xleftarrow{\$} [\![e_1]\!]_\rho; y \xleftarrow{\$} [\![e_2]\!]_\rho \textbf{ in } x + y$$
- $[\![e_1 \doteq e_2]\!]_\rho$ is defined by the clause
$$\textbf{let } x \xleftarrow{\$} [\![e_1]\!]_\rho; y \xleftarrow{\$} [\![e_2]\!]_\rho \textbf{ in } x = y$$
- $[\![\mathsf{let}\ r \xleftarrow{\$} U\ \mathsf{in}\ e]\!]_\rho$ is defined by the clause
$$\textbf{let } x \xleftarrow{\$} \mathcal{U}_{[\![U]\!]} \textbf{ in } [\![e]\!]_{\rho(r:=x)}$$

where $\mathcal{U}_{[\![U]\!]}$ denotes the uniform distribution over $[\![U]\!]$ (which is a finite set). Note that the interpretation of $[\![\mathsf{let}\ r \xleftarrow{\$} U\ \mathsf{in}\ e]\!]_\rho$ can be written in the equivalent form

$$\lambda u. \frac{\#\{x \in [\![U]\!] \mid [\![e]\!]_{\rho(r:=x)} = u\}}{\#[\![U]\!]}$$

where $\#S$ denotes the cardinality of the finite set S.

In order to illustrate the semantics of programs, consider the simple example of the closed boolean expression

$$(\mathsf{let}\ r \xleftarrow{\$}\ \mathbb{Z}_n\ \mathsf{in}\ r) \doteq 1$$

We have

$$\begin{array}{rcl} [\![(\mathsf{let}\ r \xleftarrow{\$} \mathbb{Z}_n\ \mathsf{in}\ r) \doteq 1]\!] & = & \textbf{let } x \xleftarrow{\$} \mathcal{U}_{[\![\mathbb{Z}_n]\!]} \textbf{ in } x = 1 \\ & = & \frac{1}{\#[\![\mathbb{Z}_n]\!]} \\ & = & \frac{1}{[\![n]\!]} \end{array}$$

B. Batch verification

We extend to arbitrary boolean-valued expressions the notion of batch verifier introduced in Section III. Let e and e' be two boolean-valued expressions. Assume $\mathrm{FV}(e) = X \cup \{x\}$ and $\mathrm{FV}(e') = X \cup \{y\}$ where $x : U$ and $y : U^m$. We say that e' is an ϵ-correct batch expression for e w.r.t. the mapping $x \mapsto y$, written $\vDash e \Rightarrow^{\epsilon}_{x \mapsto y} e'$, iff for every interpretation ρ and tuple of values $\vec{v}$,

$$0 \leq \Pr\left[[\![e']\!]_{\rho(y=\vec{v})} = \mathsf{tt}\right] - \Pr\left[\bigwedge_{i<n} [\![e]\!]_{\rho(x=\vec{v}_{[i]})} = \mathsf{tt}\right] \leq \epsilon$$

To capture the batch security definition for signatures from Section III-C, we can use the mapping $w \mapsto \mathbf{w}$ for variables $w : U$ and $\mathbf{w} : U^m$ where U denotes the type of triples of public keys, messages, and signatures. To fix the public key, we can remove the public key from the type U and use a shared free variable for the public key in both expressions.

In the remainder of the section, we derive valid batch verifiers for algorithms that operate on prime order groups or composite order groups whose order has no small prime factors. We proceed in two steps; first, we apply the Small Exponent Test to the original verifier; then, we perform semantics-preserving transformations.

C. Small Exponent Test for composite order groups

We now formulate a more general Small Exponent Test that applies to groups whose order has no small prime factors.

Proposition IV.1 (Small Exponent Test) Let $n = p_1 \cdots p_k$ denote the group order and let $a_i, b_i \in \mathbb{Z}_n$ for $i < m$. Consider the expressions $s = \sum_{i<m}[x_i]a_i$ and $t = \sum_{i<m}[x_i]b_i$, where x_i are chosen uniformly at random over $(1 \ldots 2^\ell)$. Then the probability that $s = t$ with $a_i \neq b_i$ for some i is upper bounded by $1/p + 2^{-\ell}$ where $p = \min\{p_1, \ldots, p_k\}$. In the common case where $2^l < p$, the probability is upper bounded by $2^{-\ell}$.

Proof sketch: We first consider the prime-order case for clarity. If $s = t$ with $a_i \neq b_i$, then there is a c that is independent of x_i such that $x_i\,(a_i - b_i) = c$. Since $a_i - b_i \neq 0$ in $\mathbb{Z}_p$, we can divide c by $a_i - b_i$ in $\mathbb{Z}_p$ and conclude that the probability is equal to $2^{-\ell}$ since x_i is sampled independently from $c/(a_i - b_i)$.

In the composite case, it is possible that $a_i - b_i \neq 0$ in $\mathbb{Z}_n$, but $a_i - b_i = 0$ modulo some prime factor p of n. Hence $a_i - b_i$ is not invertible in $\mathbb{Z}_n$ and the reasoning from the prime-order case is not directly applicable. Nevertheless, there must be some other prime factor p' of n such that $a_i - b_i \neq 0$ modulo p' and therefore invertible in $\mathbb{Z}_{p'}$. Hence, we can reduce both sides of the equation modulo p' and continue as in the prime order case. Concretely, there is some value d that is independent of x_i such that $x_i \mod p' = d$. If $2^l < p'$ then $x_i \mod p' = x$ is still uniform in 2^l and we obtain the bound 2^{-l}. Otherwise, the maximum probability weight in the distribution defined by sampling uniformly from $1, \ldots, 2^\ell$ and reducing modulo p' is upper-bounded by $2^{-l} + 1/p'$. ■

Note that for the case of $k = 2$, Blazy et al. [14] obtain the bound $\max\{p_1, p_2\}\,2^{-l}$ which requires 2^l to be significantly larger than the prime factors of the group order.

Given a boolean-valued expression e, we write $e \Rightarrow^{\mathrm{SmallExps}(\ell)}_{x \mapsto y} e'$ if e' is obtained from e by applying the small exponent step to check $e\{y_i/x\}$ for all valid indexes of y.

D. Equational reasoning

Fig. 4 presents a set of axioms to derive valid equalities in all interpretations. One can derive from these axioms further equations using the standard rules of equational logic: reflexivity, symmetry, transitivity, congruence, and substitution.

Proposition IV.2 If $e \equiv e'$ is derivable from axioms using equational rules, then $\vDash e \equiv e'$, i.e. for all interpretations and valuations ρ, $[\![e]\!]_\rho$ and $[\![e']\!]_\rho$ denote the same distributions.

One key step to verify automatically the batch verifiers of AutoBatch is to be able to check efficiently whether two

expressions are provably equivalent using these rules. Most techniques used by AutoBatch, to the notable exception of the Small Exponent Test which is addressed below, can be justified in this equational setting. In many cases, such as moving scalars inside pairings (Technique 2 of [5]), moving sums inside pairings (Technique 3 of [5]), or distribute sums (Technique 5 of [5]), the technique is a simple application of our rules. A more interesting example is Technique 4 of [5], which is used to optimize the Waters Hash, and consists in replacing

$$\sum_{i<m} \mathsf{e}(a_i, \sum_{j<m'} [u_{i,j}]b_j)$$

by

$$\sum_{j<m'} \mathsf{e}(\sum_{i<m} [u_{i,j}]a_i, b_j)$$

to reduce the required pairing computations if m' is less than m. This step can be justified as follows; first, move the big sum out of the pairing in the first expression, then move the scalar out of the pairing. One obtains an expression

$$\sum_{i<m} \sum_{j<m'} [u_{i,j}]\mathsf{e}(a_i, b_j)$$

By performing the same steps on the second expression, one obtains an expression

$$\sum_{j<m'} \sum_{i<m} [u_{i,j}]\mathsf{e}(a_i, b_j)$$

The two expressions are provably equivalent using the last rule for Abelian groups; since all steps preserve equality, the two original expressions are equivalent.

E. Correctness of batch verifiers

The correctness of batch verifiers is a direct consequence of the previous results.

Proposition IV.3 If $\vDash e \Rightarrow^{\mathrm{SmallExps}(\ell)}_{x \mapsto y} e'$ and $\vDash e' \equiv e''$ then $\vDash e \Rightarrow^{2^{-\ell}}_{x \mapsto y} e''$.

F. Implementation in EasyCrypt

We have implemented an automated procedure to prove the correctness of batch verifiers in EasyCrypt. Given a boolean expression e_0 and a candidate batch expression e_1 for e_0, we first build the naive batch verifier for e_0 and apply the Small Exponent Test to obtain a next expression e_2. Our goal is then to prove that $e_1 \doteq e_2$.

The first step is to normalize the equation by first pulling at the top all random samplings and then reducing all deterministic let bindings. This leaves us with a sequence of random samplings followed by an equation of the form $e'_1 \doteq e'_2$, where the e'_i's do not contain any (deterministic or random) let binding. We are left to decide the validity of $e'_1 \doteq e'_2$.

We first inject the equation in $\mathbb{Z}_n$ by applying the discrete logarithm, reducing the problem to $\log(e'_1) \doteq \log(e'_2)$. Then, we use the EasyCrypt rewrite engine to normalize the $\log(e'_i)$'s. During this step we apply morphism equalities and distribution laws, and we push big operators, log and (scalar) multiplication to the leaves of the $\log(e'_i)$'s. Besides the standard equalities of Figure 4, log obeys the common group morphism equalities augmented by

$$\forall e_1, e_2.\ \log(\mathsf{e}(e_1, e_2)) \doteq \log(e_1) + \log(e_2)$$

This last property is crucial for reducing an equality in the target group into an equality in the ring of scalars. Eventually, we obtain an equation of the form

$$e_1 + \cdots + e_n \doteq e_{n+1} + \cdots e_{n+k}$$

where each e_i is a nested sum of big operators and each summand is a product, i.e. is of the form

$$\sum_{j_1<k_1} \cdots \sum_{j_s<k_s} e''_1 \cdot e''_2 \cdots e''_n$$

We then reorder top-level big sums of the $e_{n+1}, \ldots, e_{n+k}$'s. This reordering is again certified by the EasyCrypt rewrite engine.

We decide the latter equality by using a mixture of a free ring expression simplifier — part of EasyCrypt and that has been obtained by extraction from a certified Coq program — and congruence. Starting from a possibly impure ring expression (i.e. containing symbols outside of the ring theory — at this stage, big operators are no more considered as part of the ring signature), we gather the maximal impure subterms. We then abstract all these impure terms by fresh variables, using the same variable for subterms that can be (recursively) equated by the procedure. The obtained expression is thus a pure ring expression that can be normalized by the core EasyCrypt ring simplifier.

In turn, terms headed by a non-ring symbol are equated by normalizing them and checking for syntactical equality. Normalizing such a term amounts to normalize all its subterms headed by a ring symbol, using the procedure just described in the previous paragraph.

Finally, the latter equation is accepted if the normalization of $e_1 + \cdots + e_n$ and $e_{n+1} + \cdots + e_{n+k}$ leads to the same term. In that case, considering that all the equations of Figure 4 extended by the ones related to log have been formally verified in EasyCrypt, the proof of correctness of the batch verifier only relies on the core rewrite engine of EasyCrypt and its certified ring normalizer.

V. Applications to batch signature verification

To measure the effectiveness of our approach, we evaluate our techniques on several existing signature schemes from the literature. More precisely, we verified with EasyCrypt most of the batch verifiers produced by AutoBatch in [5]. As a result, our techniques extend to several signature types in the literature including identity-based signatures, group signatures, and ring signatures.

In Figure 5, we show all the batch verifiers output by AutoBatch certified by EasyCrypt. Additionally, we verified the Waters09 [57] batch result which is known to include a negligible correctness error in the individual verification. As discussed in the extended work on AutoBatch [6], it was an open question whether the batch verification security definition can be met by such schemes. EasyCrypt handles such

Abelian group $(+,-,0)$ for $\mathbb{G}_i$ and $\mathbb{Z}_n$:

$$\begin{aligned} a+(b+c) &\equiv (a+b)+c \\ a+b &\equiv b+a \\ a+(-a) &\equiv 0 \quad \text{if } \mathsf{det}(a) \\ a+0 &\equiv a \\ \sum_{i<m}(a+b) &\equiv (\sum_{i<m} a)+(\sum_{i<m} b) \\ \sum_{i<m}(\sum_{j<m'} a) &\equiv \sum_{j<m'}(\sum_{i<m} a) \end{aligned}$$

Commutative monoid $(\cdot,1)$ in $\mathbb{Z}_n$:

$$\begin{aligned} u\cdot(v\cdot w) &\equiv (u\cdot v)\cdot w \\ u\cdot v &\equiv v\cdot u \\ u\cdot 1 &\equiv u \end{aligned}$$

Logical rules:

$$\begin{aligned} \bigwedge_{i<m}(\phi\wedge\psi) &\equiv (\bigwedge_{i<m}\phi)\wedge(\bigwedge_{i<m}\psi) \\ \bigwedge_{i<m}(\bigwedge_{j<m'}\phi) &\equiv \bigwedge_{j<m'}(\bigwedge_{i<m}\phi) \\ \bigwedge_{i<m} a &\equiv a \quad \text{if } \mathsf{det}(ax) \text{ and } i\notin \mathrm{FV}(a) \\ a\wedge a &\equiv a \quad \text{if } \mathsf{det}(a) \\ a\doteq b &\equiv a-b\doteq 0 \\ 0\doteq 0\wedge u &\equiv u \end{aligned}$$

Let binding:

$$\mathsf{let}\ x\leftarrow e_1\ \mathsf{in}\ e_2 \equiv e_2\{e_1/x\} \quad \text{if } \mathsf{det}(e_1)$$

Distributivity in $\mathbb{Z}_n$:

$$\begin{aligned} u\cdot(v+w) &\equiv u\cdot v+u\cdot w \\ u\cdot(\sum_{i<m} v) &\equiv \sum_{i<m} u\cdot v \end{aligned}$$

Scalar multiplication:

$$\begin{aligned} [u][v]c &\equiv [u\cdot v]c \\ [1]c &\equiv c \\ [u+v]a &\equiv [u]a+[v]a \\ [u]a+[u]b &\equiv [u](a+b) \\ [\sum_{i<m} u]a &\equiv \sum_{i<m}[u]a \quad \text{if } i\notin \mathrm{FV}(a) \\ [u](\sum_{i<m} a) &\equiv \sum_{i<m}[u]a \quad \text{if } i\notin \mathrm{FV}(u) \end{aligned}$$

Bilinear map:

$$\begin{aligned} \mathsf{e}(a+b,c) &\equiv \mathsf{e}(a,c)+\mathsf{e}(b,c) \\ \mathsf{e}(a,b+c) &\equiv \mathsf{e}(a,b)+\mathsf{e}(a,c) \\ \mathsf{e}([a]b,c)) &\equiv [a](\mathsf{e}(b,c)) \\ \mathsf{e}(a,[b]c)) &\equiv [b](\mathsf{e}(a,c)) \\ \sum_{i<m}\mathsf{e}(a,b) &\equiv \mathsf{e}(\sum_{i<m} a,b) \quad \text{if } i\notin \mathrm{FV}(b) \\ \sum_{i<m}\mathsf{e}(a,b) &\equiv \mathsf{e}(a,\sum_{i<m} b) \quad \text{if } i\notin \mathrm{FV}(a) \end{aligned}$$

Sampling:

$$\begin{aligned} \bigwedge_{i<m}(\mathsf{let}\ r\xleftarrow{\$} U\ \mathsf{in}\ a) &\equiv \mathsf{let}\ r\xleftarrow{\$} U^m\ \mathsf{in}\ (\bigwedge_{i<m} a\{r_{[i]}/r\}) \\ \sum_{i<m}(\mathsf{let}\ r\xleftarrow{\$} U\ \mathsf{in}\ a) &\equiv \mathsf{let}\ r\xleftarrow{\$} U^m\ \mathsf{in}\ (\sum_{i<m} a\{r_{[i]}/r\}) \end{aligned}$$

Fig. 4. Equational axioms

schemes, using our generalized definition, and certified this result from AutoBatch. We also test a variant of Waters09 [54] adapted by Abe *et al.* [1]. AutoBatch generated a batch verifier that reduces verification to 11 pairings total from $12n$ pairings in individual verification (where n = number of signatures). This result was also certified by EasyCrypt and presented in Figure 5.

VI. Applications to NIZK proofs

Zero-knowledge proofs are two-party cryptographic protocols in which a prover convinces a verifier of the validity of a statement, but does not provide the verifier any further information beyond this; such proofs can be used in particular to prove knowledge of a value that satisfies a given property, without revealing anything about this value—except for its existence and knowledge by the prover. More precisely, the properties of zero-knowledge protocols are captured by three notions: soundness, completeness, and zero-knowledge (there are in fact many variants of these notions). Informally, soundness ensures that a cheating prover cannot convince the verifier; completeness ensures that an honest prover with knowledge of a witness can always convince the verifier; zero knowledge ensures that the verifier does not learn anything from the proof, except for its validity. We refer to the literature on zero-knowledge proofs or to standard textbooks on cryptography for formal definitions. A related notion is witness indistinguishability. In many instances of zero-knowledge protocols, the prover uses a witness to prove the validity of the statement. Witness indistinguishability ensures that the proof does not reveal to the verifier which witness was used by the prover.

Zero-knowledge and witness-indistinguishable proofs were introduced around twenty-five years ago in landmarks articles, respectively by Goldwasser, Micali and Rackoff [37] and by Feige and Shamir [32] (see [36] for a survey). These proofs are useful for numerous applications, including anonymous authentication, blind signatures, electronic cash, and electronic voting. However, for practical purposes, it is generally important that zero-knowledge proofs are efficient, non-interactive protocols. The study of non-interactive zero-knowledge proofs was initiated by Blum, Feldman and Micali [15] shortly after the introduction of zero-knowledge

Scheme	Batch Verification Equation output by AutoBatch
Signatures	
BLS [18] (ss)	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]h_i, pk) \doteq \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]sig_i, g)$
CHP [21] (same time period)	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]sig_i, g) \doteq \mathsf{e}(a, \sum_{i=1}^{\eta}[\delta_i]pk_i) + \mathsf{e}(h, \sum_{i=1}^{\eta}[b_i \cdot \delta_i]pk_i)$
HW [42] (ss)	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]\sigma_{1i}, g) \doteq [\sum_{i=1}^{\eta} M_i \cdot \delta_i]U + [\sum_{i=1}^{\eta} r_i \cdot \delta_i]V + [\sum_{i=1}^{\eta} \delta_i]D$ $+\mathsf{e}(\sum_{i=1}^{\eta}[\lg(x_i) \cdot \delta_i]\sigma_{2i}, w) + \mathsf{e}(\sum_{i=1}^{\eta}[i_i \cdot \delta_i]\sigma_{2i}, i_2) + \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]\sigma_{2i}, h)$
Waters09 [57] (ss)	$\mathsf{e}([b]g_1, \sum_{i=1}^{\eta}[s_i \cdot \delta_i]\sigma_{i,1}) + \mathsf{e}([b \cdot a_1]g_1, \sum_{i=1}^{\eta}[s_{i,1} \cdot \delta_i]\sigma_{i,2})$ $+\mathsf{e}([a_1]g_1, \sum_{i=1}^{\eta}[s_{i,1} \cdot \delta_i]\sigma_{i,3}) + \mathsf{e}([b \cdot a_2]g_1, \sum_{i=1}^{\eta}[s_{i,2} \cdot \delta_i]\sigma_{i,4})$ $+\mathsf{e}({g_1}^{a_2}, \sum_{i=1}^{\eta}[s_{i,2} \cdot \delta_i]\sigma_{i,5}) \doteq \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i \cdot s_{i,1}]\sigma_{i,6}, \tau_1)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i \cdot s_{i,2}]\sigma_{i,6}, \tau_2) + \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i \cdot s_{i,1}]\sigma_{i,7}, [b]\tau_1)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i \cdot s_{i,2}]\sigma_{i,7}, [b]\tau_2) + \mathsf{e}(\sum_{i=1}^{\eta}[(\delta_i \cdot -t_i + \theta_i \cdot \delta_i \cdot tag_{i,c} \cdot t_i)]\sigma_{i,7}, w)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[\theta_i \cdot \delta_i \cdot M_i \cdot t_i]\sigma_{i,7}, u) + \mathsf{e}(\sum_{i=1}^{\eta}[\theta_i \cdot \delta_i \cdot t_i]\sigma_{i,7}, h)$ $+\mathsf{e}(g_1, \sum_{i=1}^{\eta}[-t_i \cdot \theta_i \cdot \delta_i]\sigma_{i,K}) + [\sum_{i=1}^{\eta} s_{i,2} \cdot \delta_i]A$
Waters09 variant [1, §6.4] (ss)	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_{i,2}]\sigma_{i,4}, \bar{R}) + \mathsf{e}(\sum_{i=1}^{\eta}[\delta_{i,2}]\sigma_{i,5}, [b]R) + [\sum_{i=1}^{\eta} \delta_{i,2}]P_0$ $+\mathsf{e}(\sum_{i=1}^{\eta}[-\delta_{i,2}]\sigma_{i,1}, [b]g_2) + \mathsf{e}(\sum_{i=1}^{\eta}[-\delta_{i,2}]\sigma_{i,2}, [b \cdot a]g_2)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[-\delta_{i,2}]\sigma_{i,3}, [a]g_2) + \mathsf{e}(\sum_{i=1}^{\eta}[-\delta_{i,1} \cdot m_i]\sigma_{i,5}, U_2)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[-\delta_{i,1}]\sigma_{i,5}, \bar{H}) + \mathsf{e}(g_1, \sum_{i=1}^{\eta}[\delta_{i,1}]\sigma_{i,0})$ $\doteq \mathsf{e}(U_1, [\sum_{i=1}^{\eta} m_i \cdot \delta_{i,3}]\bar{F}_2 + [\sum_{i=1}^{\eta} -m_i \cdot \delta_{i,4}]\bar{F}_3)$ $+\mathsf{e}(\hat{F}_2, [\sum_{i=1}^{\eta} m_i \cdot -\delta_{i,3}]U_2) + \mathsf{e}(\hat{F}_3, [\sum_{i=1}^{\eta} m_i \cdot \delta_{i,4}]U_2)$
CL [22] (ss)	$\mathsf{e}(g, \sum_{i=1}^{\eta}[\delta_{i,1}]b_i + [\delta_{i,2}]c_i) + \mathsf{e}(Y, \sum_{i=1}^{\eta}[-\delta_{i,1}]a_i)$ $\doteq \mathsf{e}(X, \sum_{i=1}^{\eta}[\delta_{i,2}]a_i + [m_i \cdot \delta_{i,2}]b_i)$
ID-based Signatures	
Hess [41]	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{2i}, g_2) \doteq \mathsf{e}(\sum_{i=1}^{\eta}[a_i \cdot \delta_i]pk_i, P_{pub}) + \sum_{i=1}^{\eta}[\delta_i]S_{1i}$
ChCh [25]	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{2i}, g_2) \doteq \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i](S_{1i} + [a_i]pk), P_{pub})$
Waters05 [56]	$\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{1i}, g_2) + \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{2i}, u'_1)$ $+\sum_{j=1}^{l} \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i \cdot k_{j,i}]S_{2i} + [\delta_i \cdot m_{j,i}]S_{3i}, \hat{u}_j)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{3i}, u'_2) \doteq [\sum_{i=1}^{\eta} \delta_i]B$
Group, Ring, ID-based Ring Sigs & Groth-Sahai NIZK	
BBS [16]	$\mathsf{e}((\sum_{i=1}^{\eta}[s_{i,x} \cdot \delta_i]T_{i,3} + [(-s_{i,\gamma 1} - s_{i,\gamma 2}) \cdot \delta_i]h + [-c_i \cdot \delta_i]g_1), g_2)$ $+\mathsf{e}(([\sum_{i=1}^{\eta}(-s_{i,\alpha} - s_{i,\beta}) \cdot \delta_i]h + \sum_{i=1}^{\eta}[c_i \cdot \delta_i]T_{i,3}), w) \doteq \sum_{i=1}^{\eta}[\delta_i]R_{i,3}$
Boyen [20] (same ring)	$\sum_{y=1}^{l} \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_{y,i}, \hat{A}_y) + \mathsf{e}(\sum_{i=1}^{\eta}[m_{y,i} \cdot \delta_i]S_{y,i}, \hat{B}_y)$ $+\mathsf{e}(\sum_{i=1}^{\eta}[t_{y,i} \cdot \delta_i]S_{y,i}, \hat{C}_y) \doteq \sum_{i=1}^{\eta}[\delta_i]D$
CYH [28]	$\mathsf{e}(\sum_{i=1}^{\eta}\sum_{y=1}^{l}[\delta_i](u_{y,i} + [h_{y,i}]pk_{y,i}), P) \doteq \mathsf{e}(\sum_{i=1}^{\eta}[\delta_i]S_i, g)$
Groth-Sahai NIZK [14, §5.1]	$\sum_{j=1}^{\mu} e([r_{1,1}]\sum_{i=1}^{\eta}[\alpha_{i,j}]c_{1,i} + [r_{2,1}](A_j + \sum_{i=1}^{\eta}[\alpha_{i,j}]c_{2,i}), d_{1,j})$ $+\sum_{j=1}^{\mu} e([r_{1,2}]\sum_{i=1}^{\eta}[\alpha_{i,j}]c_{1,i} + [r_{2,2}](A_j + \sum_{i=1}^{\eta}[\alpha_{i,j}]c_{2,i}), d_{2,j})$ $+\sum_{i=1}^{\eta} e([r_{1,2}]c_{1,i} + [r_{2,2}]c_{2,i}, B_i)$ $\doteq e([r_{1,1}]u_{1,1} + [r_{2,1}]u_{1,2}, \pi_{1,1}) + e([r_{1,1}]u_{2,1} + [r_{2,1}]u_{2,2}, \pi_{2,1})$ $+e([r_{1,1}]\theta_{1,1} + [r_{2,1}]\theta_{1,2}, v_{1,1}) + e([r_{1,1}]\theta_{2,1} + [r_{2,1}]\theta_{2,2}, v_{2,1})$ $+e([r_{1,2}]u_{1,1} + [r_{2,2}]u_{1,2}, \pi_{1,2}) + e([r_{1,2}]u_{2,1} + [r_{2,2}]u_{2,2}, \pi_{2,2})$ $+e([r_{1,2}]\theta_{1,1} + [r_{2,2}]\theta_{1,2}, v_{1,2}) + e([r_{1,2}]\theta_{2,1} + [r_{2,2}]\theta_{2,2}, v_{2,2}) + [r_{2,2}]T$

ss = same signer

Fig. 5. These are the final batch verification equations output by AutoBatch and verified by EasyCrypt. Due to space, we do not include the full schemes or further describe the elements of the signature or our shorthand for them, such as setting $h = H(M)$ in BLS. We briefly mention the Waters09 variant and Waters05. P_0 represents the precomputed pairing $\mathsf{e}({g_1}^{\rho}, {g_2}^{(\alpha \cdot b)/\rho})$. Similarly, for Waters05, $B = \mathsf{e}(g_1, g_2)$.

proofs, but it has proved remarkably difficult to achieve efficient non-interactive zero-knowledge (NIZK) and and non-interactive witness-indistinguishable (NIWI) proofs for a general class of statements.

In 2008, Groth and Sahai [38] developed new techniques that allow to build efficient NIZK and NIWI proofs for proving the satisfiability of pairing-based statements. Statements are of one of the following forms, where x_i and y_j are the parameters of the statements (and the other constituents are known public values):

- equations in the target group (*i.e.*, pairing-product equations):

$$\sum_{i<m} \mathsf{e}(a_i, x_i) + \sum_{j<m'} \mathsf{e}(y_j, b_j) + \sum_{i<m} \sum_{j<m'} [u_{i,j}]\mathsf{e}(x_i, y_j) = c$$

  where $u_{i,j} \in \mathbb{Z}_n$, $a_i, y_j \in \mathbb{G}_1$, $b_j, x_i \in \mathbb{G}_2$ and $c \in \mathbb{G}_T$;
- equations in the first source group (*i.e.*, multi-scalar multiplication equations), the case of the second source group is dual:

$$\sum_{i<m} [x_i]a_i + \sum_{j<m'} [v_j]b_j + \sum_{i<m} \sum_{j<m'} [u_{i,j}y_j]b = c$$

  where $u_{i,j}, x_i, y_j, v_j \in \mathbb{Z}_n$, and $a_i, b_i, c \in \mathbb{G}_1$;
- equations in $\mathbb{Z}_n$ (*i.e.* quadratic equations):

$$\sum_{i<m} x_i u_i + \sum_{j<m'} y_j v_j + \sum_{i<m} \sum_{j<m'} x_i\, y_j\, u_{i,j} = w$$

  where $u_{i,j}, x_i, y_j, v_j, w \in \mathbb{Z}_n$.

They also provide three instantiations of their methods. The first instantiation is in the symmetric setting for composite order groups where the order is a RSA modulus, i.e. the product of two distinct primes, and is based on the subgroup decision SD assumption. The second instantiation is in the asymmetric setting for prime order groups and is based on the SXDH assumption. Finally, the third instantiation is in the symmetric setting for prime order groups and is based on the DLIN assumption. For each case, one obtains a set of equations to verify.

In a follow-up work, Blazy et al. [14] apply tools from batch verification to improve the efficiency of checking NIZK and NIWI proofs for the satisfiability of the aforementioned statements. Their primary focus is on SXDH and DLIN since the instantiation based the subgroup decision assumption is the least practical. We present an extension of our framework that is able to automatically compute and certify batch Groth-Sahai proofs in the SXDH instantiation.

First, we apply our framework to the problem of batching pairing-product equations presented in Section 5.1 of Blazy et al. [14]. Here, AutoBatch automatically computes the batch verifier shown in Figure 5 which is equivalent to the batch verifier that Blazy et al. obtain manually. We then use our EasyCrypt backend to confirm the correctness of the computed batch verifier.

Afterwards, we use our EasyCrypt backend to verify the correctness of the batch verifier for multi-scalar equations presented in Section 5.2 of Blazy et al. [14]. After applying the Small Exponent Test to the naive batch verifier, EasyCrypt fails to prove its equivalence to the batch verifier proposed by Blazy. Looking more carefully at their definition, we find an error in the right-hand side of the equation, the term

$$\begin{aligned}
&\mathsf{e}([r_{1,1}]\theta_1 + [r_{2,1}]\theta_2, v_{1,1}) + \mathsf{e}([\underline{r_{2,1}}]\theta_1 + [r_{2,2}]\theta_2, v_{1,2}) + \\
&\mathsf{e}([r_{1,1}]u_{1,1} + [r_{2,1}]u_{1,2}, \pi_{1,1}) + \mathsf{e}([r_{1,1}]u_{2,1} + [r_{2,1}]u_{2,2}, \pi_{2,1}) + \\
&\mathsf{e}([r_{1,2}]u_{2,1} + [r_{2,2}]u_{2,2}, \pi_{1,2}) + \mathsf{e}([\underline{r_{2,1}}]u_{2,1} + [r_{2,2}]u_{2,2}, \pi_{2,2}) + \\
&\mathsf{e}(T_1, [r_{2,1}]v_{2,1} + [r_{2,2}](g_2 + v_{2,2})
\end{aligned}$$

should be

$$\begin{aligned}
&\mathsf{e}([r_{1,1}]\theta_1 + [r_{2,1}]\theta_2, v_{1,1}) + \mathsf{e}([\underline{r_{1,2}}]\theta_1 + [r_{2,2}]\theta_2, v_{1,2}) + \\
&\mathsf{e}([r_{1,1}]u_{1,1} + [r_{2,1}]u_{1,2}, \pi_{1,1}) + \mathsf{e}([r_{1,1}]u_{2,1} + [r_{2,1}]u_{2,2}, \pi_{2,1}) + \\
&\mathsf{e}([r_{1,2}]u_{2,1} + [r_{2,2}]u_{2,2}, \pi_{1,2}) + \mathsf{e}([\underline{r_{1,2}}]u_{2,1} + [r_{2,2}]u_{2,2}, \pi_{2,2}) + \\
&\mathsf{e}(T_1, [r_{2,1}]v_{2,1} + [r_{2,2}](g_2 + v_{2,2}).
\end{aligned}$$

In other words, the (underlined) coefficients of θ_1 in the second pairing and of $u_{2,1}$ in the sixth pairing should be $r_{1,2}$ instead of $r_{2,1}$. While the errors are certainly typos, it shows the importance of having computer-aided methods to generate and validate equations of this sort.

VII. Related work

A. Batch verification

Batch Verification has a long history spanning over two decades. Fiat introduced the notion of batching for a variant of RSA [34] in 1989, which focused on batching modular exponentiations. Naccache *et al.* [52] presented an interactive verifier for DSA that was later broken by Lim and Lee [49] in 1994. Laih *et al.* [48] proposed batch verifiers for DSA/RSA signatures, but the RSA batch verifier was later broken by Boyd and Pavlovski [19]. Additional batch verification techniques were proposed for RSA and DSA by others [39], [40], but sadly they were trivially broken [19], [43], [44]. As indicated before, Bellare *et al.* [11] proposed three generic methods in 1998 for securely batching modular exponentiations, one of which is the small exponents test.

Several subsequent batch verification proposals [25], [58], [60], [61] extended these methods to support batching pairings, but most of them were found to be flawed. The issues include in some cases insufficient or absent proof of correctness, incorrect application of the small exponents test or invalid assumptions on the algebraic structure of certain signature elements. See [23], [55] for more details.

More recently, Ferrera *et al.* [33] demonstrate several general techniques for securely batching pairing-based signatures and show several batch constructions from the literature (*e.g.* group and ring-signatures, etc). Akinyele *et al.* [5] further build on this work in AutoBatch by automating the application of the batching techniques in addition to automating the search for an optimized batch verifier. The tool proves effective in automatically rediscovering existing lower bounds for signatures from the literature and in some cases, outperform existing results. As indicated before, AutoBatch produces a human-readable proof of correctness which is not machine-checked. This work develops a systematic approach using EasyCrypt to certify the correctness and security of batch verifiers produced by AutoBatch.

B. Computer-assisted cryptographic proofs and design

There has been significant work in developing computer-assisted frameworks for analyzing cryptographic constructions. Most of this work has concentrated on the symbolic model of cryptography [13], in which cryptographic primitives are idealized. However, more recent work has focused on developing tools to reason directly in the computational model. CryptoVerif [12] is the first such tool that was used for analyzing the security of cryptographic primitives and protocols. EasyCrypt [10] is an interactive tool based on deductive verification methods that has been used to analyze cryptographic constructions. EasyCrypt can be used as a certifying backend for cryptographic compilers: for instance, ZKCrypt [8], a verifying zero-knowledge compiler based on CertiCrypt and EasyCrypt and on the CACE compiler [7]. ZooCrypt [9] is another automated tool that allows the synthesis and analysis of public-key encryption schemes based on one-way trapdoor permutations and random oracles. ZooCrypt is similar to AutoBatch in the sense that it proposes to users new algorithms; in contrast to AutoBatch, it does not require any user input—except bounds on the size of algorithms to be synthesized. Lastly, AutoGroup and AutoStrong [4] are automated tools that use SMT solvers for the synthesis of pairing-based encryption and signature schemes based on users specific security and efficiency goals. These tools are also similar to AutoBatch in terms of searching for optimal solutions on given inputs (although AutoBatch does not provide any guarantee of optimality).

C. Equational reasoning in cryptography

There has been extensive work to develop symbolic protocol verification techniques in presence of equational theories; see for instance [29]. Bilinear pairings are considered for instance in [47], [53]. These works focus on a simpler setting; in particular, they do not consider big operators. Dougherty and Guttman [30] consider an equational theory without pairings, but with division (essentially to reason about fields, rather than rings); extending our work to account for division (in the case of prime order groups) is an important avenue for further work.

VIII. Conclusion

There is growing interest in developing computer-aided methods for cryptography. Focusing on the practically relevant example of batch verification for pairing-based signatures and proofs of knowledge, we have illustrated how two previously disconnected approaches to computer-aided cryptography (respectively focused on automated optimization and interactive proofs) naturally converge to deliver optimized *and* formally verified cryptographic algorithms.

One contribution of independent interest is our method for checking equality of pairing-based computations, which is very useful in the more general context of interactive verification in EasyCrypt and could be helpful to the cryptographic community for checking complex equalities that arise in their proofs (although their proofs are normally carried on pen-and-paper, having automated support for such mundane, error-prone computations would be beneficial). It would be interesting to extend the method to multi-linear maps, an emerging field in cryptography that involves complex algebraic manipulations and would clearly benefit from automated procedures to verify equivalence of computations based on multi-linear maps.

An avenue for further work is to extend AutoBatch into a compiler that generates implementations of batch Groth-Sahai proofs from pairing-based equations; we expect that some of the techniques developed in ZKCrypt [8] will prove useful in this setting. More generally, we intend to pursue developing computer-aided tools that assist with the design, generation and verification of cryptographic constructions, and expect that such tools will play an increasingly prominent role in developing new, efficient and secure schemes.

Ackowledgements: This research is partially supported by ONR grant N000141210914 and N00014-11-1-0470, Madrid regional project S2009TIC-1465 PROMETIDOS, and Spanish projects TIN2009-14599 DESAFIOS 10 and TIN2012-39391-C04-01 Strongsoft; the U.S. Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. The views expressed are those of he authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. Cryptology ePrint Archive, Report 2012/285, 2012. http://eprint.iacr.org/.

[2] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, 2010.

[3] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[4] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using smt solvers to automate design tasks for encryption and signature schemes. In *ACM CCS'13*, pages 399–410, New York, NY, USA, 2013. ACM.

[5] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *ACM CCS'12*, pages 474–487, New York, NY, USA, 2012. ACM.

[6] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. Cryptology ePrint Archive, Report 2013/175, 2013. http://eprint.iacr.org/.

[7] José Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. A certifying compiler for Zero-Knowledge Proofs of Knowledge based on Sigma-protocols. In *Computer Security – ESORICS 2010, 15th European Symposium on Research In Computer Security*, volume 6345 of *Lecture Notes in Computer Science*, pages 151–167, Heidelberg, 2010. Springer.

[8] José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella-Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In *19th ACM Conference on Computer and Communications Security, CCS 2012*, pages 488–500, New York, 2012. ACM.

[9] Gilles Barthe, Juan-Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella-Béguelin. Fully automated analysis of padding-based encryption in the computational model. In *ACM Conference on Computer and Communications Security*, 2013.

[10] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume

6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.

[11] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT'98*, volume 1403 of LNCS, pages 236–250. Springer-Verlag, 1998.

[12] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE Symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.

[13] Bruno Blanchet. Security protocol verification: Symbolic and computational models. In *1st International Conference on Principles of Security and Trust, POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 3–29, Heidelberg, 2012. Springer.

[14] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth-sahai. In *ACNS '10*, pages 218–235. "Springer-Verlag", 2010.

[15] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *20th Annual ACM Symposium on Theory of computing, STOC 1988*, pages 103–112. ACM, 1988.

[16] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, volume 3152 of LNCS, pages 45–55, 2004.

[17] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. extended abstract in Crypto'01.

[18] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01*, volume 2248 of *Lectures Notes in Computer Science*, pages 514–532, 2001.

[19] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology – ASIACRYPT '00*, volume 1976, pages 58–71, 2000.

[20] Xavier Boyen. Mesh signatures: How to leak a secret with unwitting and unwilling participants. In *EUROCRYPT, volume 4515 of LNCS*, pages 210–227. Springer, 2007.

[21] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In *EuroCrypt'07*, volume 4515 of LNCS, pages 246–263. Springer-Verlag, 2007. Full version at http://eprint.iacr.org/2007/172.

[22] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[23] Tianjie Cao, Dongdai Lin, and Rui Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.

[24] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, pages 680–699, 2013.

[25] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In *PKC '03*, volume 2567 of LNCS, pages 18–30. Springer-Verlag, 2003.

[26] Sanjit Chatterjee and Palash Sarkar. HIBE with short public parameters without random oracle. In *ASIACRYPT '06*, volume 4284 of LNCS, pages 145–160, 2006.

[27] Jung Hee Cheon and Dong Hoon Lee. Use of sparse and/or complex exponents in batch verification of exponentiations. *IEEE Trans. Computers*, 55(12):1536–1542, 2006.

[28] Sherman S. M. Chow, Siu-Ming Yiu, and Lucas C.K. Hui. Efficient identity based ring signature. In *ACNS*, volume 3531 of LNCS, pages 499–512, 2005.

[29] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[30] Daniel J. Dougherty and Joshua D. Guttman. An algebra for symbolic diffie-hellman protocol analysis. In Catuscia Palamidessi and Mark Ryan, editors, *Trustworthy Global Computing*, volume 8191 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2013.

[31] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for diffie-hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 129–147. Springer, 2013.

[32] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426. ACM, 1990.

[33] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *CT-RSA*, volume 5473 of LNCS, pages 309–324, 2009.

[34] Amos Fiat. Batch RSA. In *Advances in Cryptology – CRYPTO '89*, volume 435, pages 175–185, 1989.

[35] Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William S. Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 2004.

[36] O. Goldreich. Zero-knowledge twenty years after its invention. Technical Report TR02-063, Electronic Colloquium on Computational Complexity, 2002.

[37] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[38] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology–EUROCRYPT 2008*, pages 415–432. Springer, 2008.

[39] Lein Harn. Batch verifying multiple DSA digital signatures. *Electronics Letters*, 34(9):870–871, 1998.

[40] Lein Harn. Batch verifying multiple RSA digital signatures. *Electronics Letters*, 34(12):1219–1220, 1998.

[41] Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, volume 2595 of LNCS, pages 310–324. Springer-Verlag, 2002.

[42] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *Advances in Cryptology – EUROCRYPT*, 2009.

[43] Min-Shiang Hwang, Cheng-Chi Lee, and Yuan-Liang Tang. Two simple batch verifying multiple digital signatures. In *3rd Information and Communications Security (ICICS)*, pages 233–237, 2001.

[44] Min-Shiang Hwang, Iuon-Chang Lin, and Kuo-Feng Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences*, 11(1):15–19, 2000.

[45] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *Algorithmic Number Theory, ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[46] Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive nizk proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT*, volume 8269 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[47] Steve Kremer and Laurent Mazaré. Computationally sound analysis of protocols using bilinear pairings. *Journal of Computer Security*, 18(6):999–1033, 2010.

[48] Chi-Sung Laih and Sung-Ming Yen. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.

[49] C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), pages 1592–1593, 1994.

[50] Anna Lysyanskaya, Ronald L Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, pages 184–199. Springer, 2000.

[51] D. Naccache. Secure and *practical* identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.

[52] David Naccache, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli. Can DSA be improved? complexity trade-offs with the digital signature standard. In *Advances in Cryptology – EUROCRYPT '94*, volume 950, pages 77–85, 1994.

[53] Alisa Pankova and Peeter Laud. Symbolic analysis of cryptographic protocols containing bilinear pairings. In *Computer Security Foundations Symposium (CSF)*, pages 63–77. IEEE, 2012.

[54] Somindu C. Ramanna, Sanjit Chatterjee, and Palash Sarkar. Variants of waters' dual-system primitives using asymmetric pairings. Cryptology ePrint Archive, Report 2012/024, 2012. http://eprint.iacr.org/.

[55] Martin Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.

[56] Brent Waters. Efficient Identity-Based Encryption without random oracles. In *EUROCRYPT*, volume 3494 of LNCS, pages 114–127, 2005.

[57] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. Cryptology ePrint Archive, Report 2009/385, 2009. http://eprint.iacr.org/.

[58] HyoJin Yoon, Jung Hee Cheon, and Yongdae Kim. Batch verifications with ID-based signatures. In *ICISC*, Lecture Notes in Computer Science, pages 233–248, 2004.

[59] Chenxi Zhang, Rongxing Lu, Xiaodong Lin, Pin-Han Ho, and Xuemin Shen. An efficient identity-based batch verification scheme for vehicular sensor networks. In *INFOCOM*, pages 246–250. IEEE Press, 2008.

[60] Fangguo Zhang and Kwangjo Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In *8th Information Security and Privacy, Australasian Conference (ACISP)*, volume 2727, pages 312–323, 2003.

[61] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *Progress in Cryptology – INDOCRYPT '03*, volume 2904, pages 191–204, 2003.