Batch Pairing Delegation*

Patrick P. Tsang[1], Sherman S.M. Chow[2], and Sean W. Smith[1]

[1] Department of Computer Science
Dartmouth College
Hanover, NH 03755, USA
{patrick,sws}@cs.dartmouth.edu
[2] Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY 10012, USA
schow@cs.nyu.edu

Abstract. Pairing-based cryptography (PBC) has enabled the construction of many cryptographic protocols. However, there are scenarios when PBC is too heavyweight to use, such as when the computing devices are resource-constrained. Pairing delegation introduced in [19] provides a solution by offloading the computation to more powerful entities.
In this paper, we introduce the concept of, and construct several protocols for, *batch pairing delegation*, which offers significantly improved efficiency over multiple runs of state-of-the-art (non-batch) delegation protocols. We prove the security of our proposed protocols in the model we formalized for batch pairing delegation. Also, we have implemented our protocols in software for experimentation.
Moreover, we argue that the secure delegation of pairing computation, batched or not, requires different protocols depending on the semantic meaning of the pairings. We propose a taxonomy that classifies pairings into *seven* types to assist in choosing the right delegation protocol.
Finally, we propose a novel application of pairing delegation in trusted computing — we show how pairing delegation can be leveraged to build a secure coprocessor for pairing computation more cost-effectively.

1 Introduction

Pairing-based Cryptography. Since the first constructive uses of pairings over elliptic curves in cryptography such as tripartite key exchange [31] and identity-based encryption (IBE) [8, 36], *Pairing-Based Cryptography (PBC)* has enabled for the first time secure and efficient construction of many novel cryptographic schemes such as attribute-based encryption [4, 28], broadcast encryption [1, 11], certificateless encryption [21, 24], forward-secure encryption [14, 40], searchable

* This research was sponsored in part by the National Cyber Security Division of the Department of Homeland Security, under Grant Award Number 2006-CS-001-000001, and by the NSF, under grant CNS-0524695. The views and conclusions do not necessarily reflect the views of the sponsors.

encryption [12, 38], aggregate signatures [9], short signatures [7], perfect non-interactive zero-knowledge (NIZK) [29] and multi-theorem NIZK [16].

Pairing Delegation. Pairing delegation, first introduced by Chevallier-Mames et al. [19], is a protocol during which an entity offloads the computation of pairings to another entity. In this paper, we refer to the entity who delegates the computation as the *Delegator*, or *Ron*, and the entity who actually does the work as the *Delegatee*, or *Ellen*.[1] The fact that computationally limited devices such as smartcards are slow and resource-constrained for computing pairings has been a major motivation for pairing delegation in [19, 32]. By delegating the computation to more powerful machines such as PCs on which more efficient architecture and libraries for pairing computation exist, it becomes possible for those devices to execute various pairing-based cryptographic algorithms.

Security becomes a concern when the delegatee is not fully trusted by the delegator, and/or the two parties are communicating over an insecure channel. A pairing computation should be delegated as if the delegator computed the pairing himself. More precisely, neither the delegatee nor an eavesdropper should be able to learn anything about the pairing being computed, except perhaps the fact that the delegator is trying to compute a pairing. Furthermore, the delegator should be able to tell if he ends up with a correct answer to the computation of the pairing despite the adverse environment.

The only follow-up work (that the authors are aware of) gives several new protocol constructions with better efficiency and is due to Kang et al. [32].

Batch Processing. Under certain circumstances, processing tasks in batch rather than individually yields better efficiency. In batch signature verification [3], for instance, if verification returns valid, the verifier is assured that all signatures in the batch are valid with overwhelming probability. Otherwise, the verifier knows there is at least one bad signature. Further work on this topic studies efficient means to identify bad signatures [34, 35]. Similarly, our idea of delegating pairing computation in batch tries to improve overall efficiency by delegating in batch rather than one by one independently.

Our Contributions. We make the following contributions in this paper:

- We introduce the concept of *batch pairing delegation* and provide *four* protocol constructions, which offer better efficiency than independently invoking state-of-the-art non-batch delegation protocols. We also obtain new non-batch delegation protocols. We formalize a new security model for batch pairing delegation and prove the security of our protocols in the model.
- We observe that the security requirements for pairing delegation protocols, batch or not, depend on the types of pairings being delegated. We thus propose a taxonomy of pairings and survey the literature extensively.
- We propose a novel use of pairing delegation in the arena of trusted computing and discuss some practical issues of pairing delegation.

[1] A mnemonic guide: Ron is the delegatoR and Ellen is the delegateE.

Paper Organization. In Section 2, we classify pairings into types and illustrate each type with extensive examples. We propose a security model for batch pairing delegation protocols in Section 3. Section 4 presents our batch pairing delegation protocol constructions and evaluates their efficiency. We show how pairing delegation can be used in trusted computing, and discuss some less apparent costs of pairing delegation in Section 5. Section 6 concludes the paper.

2 A Taxonomy of Pairing Types

Definition. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two additive cyclic groups and $\mathbb{G}_T$ be a multiplicative cyclic group, all of prime order p.[2] Suppose P and Q are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. A function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a *pairing* if it satisfies the following properties:

- *(Bilinearity.)* $e(xA, yB) = e(A, B)^{xy}$ for all $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_p$.
- *(Non-degeneracy.)* $e(P, Q) \neq 1$, where 1 is the identity element in $\mathbb{G}_T$.
- *(Efficient Computability.)* $e(A, B)$ can be computed in polynomial time for all $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

Depending on the pairing, $\mathbb{G}_1$ may or may not be the same as $\mathbb{G}_2$, and homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ may or may not be efficiently computable. While these variations have implications on many cryptographic schemes [26], they are not our concern as our delegation protocols work securely irrespective to them.

Taxonomy. We can classify pairings $e(A, B)$ into 16 types according to whether each of the points A and B is public or secret, and a constant or a variable, which is governed by the semantic meaning of the points at the protocol level.[3] While the type of a pairing is relatively unimportant when the pairing is computed locally, it makes a huge difference in how the pairing should be delegated due to different security requirements and optimization possibilities.[4]

Consequently, one must devise, and prove the security of, a delegation protocol with respect to a specific pairing-type. Also, it is important to understand the type of a pairing in a cryptographic protocol in order to select a secure delegation protocol. We therefore propose a taxonomy of pairings into types. In addition, the taxonomy indicates that some pairing-types are more utilized than the others (in the literature today), which may be used as a heuristic to evaluate the impact of an efficient delegation protocol.

[2] Recently, *composite-order groups* with a bilinear map have been constructed [10] and used (e.g. [11–13, 16, 29]). In this paper, we assume groups are of prime order for simplicity, even though our results remain valid for composite-order groups.

[3] For example, the decryption algorithms of many IBE schemes [8, 27, 36] require the computation of $e(S, U)$, where S is the decryption key and U is a part of the ciphertext. Thus, S is a secret constant while U is a public variable.

[4] For instance, the delegator does not need to hide a point from the outside if the point is public. Also, it may be possible for the delegator to pre-compute some of the operations if one of the points is a constant.

Note that not all 16 types require a distinct delegation protocol. Half of them are duplicates of the rest due to the symmetry between the roles played by points A and B in the protocols.[5] Also, there is no need to delegate when both A and B are constant. As a result, we propose a taxonomy of pairings into 7 types.

Type-SV2. Pairings of this type are such that both points are a secret variable. We call this the "general" type because intuitively a secure delegation protocol for this type should also work securely if one or both of the points are instead public, and/or a constant. To the authors' best knowledge, no cryptographic construct in the literature requires the computation of Type-SV2 pairings. Nevertheless, existing works [19, 32] use delegation protocol for this general type as a basis to construct protocols for delegating pairings of other types.

Type-SVSC. Type-SVSC pairings have a secret variable point and a secret constant point. Their common usage is to prevent the leakage of partial knowledge about some secrets, exemplified in the following. Decryption in the inversion IBE scheme in [6] requires the computation of $e(A+rB, K)$, where A and B are from the ciphertext and (r, K) is the private key. When delegating the computation, one should treat $A + rB$ as a secret variable, or otherwise partial information about r would be leaked.[6] Another example is the ID-based key agreement protocol due to [18], wherein one has to compute the pairing of a secret key and a point derived from an ephemeral Diffie-Hellman secret exponent. Public knowledge of the latter point means partial information of such an exponent is leaked, which is not covered by the security guarantee in many key agreement protocols [22].

Few cryptographic schemes use Type-SVSC pairings. However, our delegation protocol for pairings of this type forms the basis for our other proposed protocols.

Type-SVPV. Pairings of this type are such that one point is a secret variable and the other is a public variable. We give several examples of their use here. In searchable encryption [12, 38], the search gateway is delegated with different trapdoor for different keywords, so a pairing of secret variables (the trapdoors) and public variables (ciphertext) will be used. In trace-and-revoke broadcast system [11], the secret variable is a temporary key derived from the private key in a way depending on the set of legitimate decryptors, while the public variable comes from the ciphertext as usual. Pairings of this type also appear in the private ciphertext validity checking in [27], wherein the secret variable is determined by a function of the private key and the ciphertext while the public variable comes from the ciphertext. A less obvious example is an ID-based key agreement protocol due to Chow and Choo [22]. One might think that the point associated with the secret key is essentially an ID-based signature due to [15] and can be made public. However, the other point is also a public variable so it is the only secret knowledge for ensuring the confidentiality of the session key.

[5] This is true even for asymmetric pairings.

[6] It is not stated explicitly in [6] whether r in private key is just an auxiliary data for decryption that is safe to be publicly known.

Finally, we note that the some computations appeared as Type-SVPV can actually be replaced by Type-PVPC (the secret variable is replaced by a public constant) if the secret variable A is chosen by the one who computes the pairing. Instead of choosing A directly, one can choose $a \in_R \mathbb{Z}_p$ and set $A = aP$ instead. The required value for the pairing can be obtained from exponentiating a Type-PVPC pairing with the secret exponent a. Examples include the signcryption step in Chow et al.'s scheme [23], and a zero-knowledge proof for the correctness of the partial decryption in Baek and Zheng's ID-based threshold decryption [2].

Type-SVPC. Pairings of this type have a secret variable point and a public constant point, and are rarely used in the cryptographic constructs nowadays. Type-SVPC pairings appear when one wants to verify the private key obtained from the key generation center. An example is the ciphertext-policy attribute-based encryption in [4], where the secret variables come from the different components of the private key. Again, similar to Type-SVPV pairings, some pairings that appear to be Type-SVPC can be treated as pairings of two constants instead, e.g. in the proving step of Kurosawa and Heng's ID-based identification protocol [33] and in the signing step of Hess's ID-based signature [30].

Type-PVSC. Type-PVSC pairings have a public variable point and a secret constant point, and commonly appear in the decryption algorithm, in which the decryption key is the secret constant and the ciphertext contributes to the public variable. Examples include IBE [8, 27, 36], certificateless encryption [21, 24], and other related encryption schemes such as attributed-based encryption [4, 28], ID-based broadcast encryption [1] and forward-secure encryption [14, 40]. Type-PVSC pairings also exist in some key agreement protocols [17, 18].

Type-PV2. Pairings of this type pair up two public variables. They are present in many cryptographic constructs. In particular, they are commonly found in verification of signature schemes and ciphertext validity checks in encryption schemes. Examples include Boldyreva's multisignature and blind signature [5], Boneh et al.'s aggregate signature and ring signature [9], Boneh and Boyen's short signature [7], Chow's verifiable pairing [20], Dodis and Yampolskiy's verifiable random function (VRF) [25], the public ciphertext validity checking in [2, 21], and Groth et al.'s witness-indistinguishable homomorphic proof commitments [29]. The multiplication of ciphertext in the doubly homomorphic encryption in [10] also uses this type of pairings. Type-PV2 pairings also appear in many cryptographic schemes involving an implicit tripartite key exchange [31]. The public variables come from the public keys and the proof/signature/ciphertext.

We note that one may treat the varying public key as a constant when batch-verifying signatures from the same signer, and hence classify the pairings as the Type-PVPC (see below) for better efficiency. However, if a point is a combination of a public key and a signature, this trick is not possible (unless a single pairing computation is split into two). An example is Cha and Cheon's scheme [15].

Type-PVPC. Pairings of this type are such that one of points is a public variable and the other is a public constant. They commonly appear in the encryption algorithm of IBE schemes based on full-domain hash [8, 40], and the verification of pairing-based schemes such as Boldyreva's multisignature and blind signature [5], Boneh et al.'s aggregate signature, ring signature and verifiably encrypted signature [9], Boyen and Water's group signature [13], Chow's verifiable pairing [20], the VRF of Chase and Lysyanskaya [16] and Dodis and Yampolskiy [25], and Hess's ID-based signature [30]. In these examples, the public constant is either the master public key or a group generator, and the public variable comes from the identity of the decryptor or the signature respectively.

Summary. We end this section by noting that pairings of type PVSC, PV2 and PVPC are widely used by cryptographers today. Delegation protocols for these types can thus be considered very applicable. As we shall see, our proposed protocols cover two of these three types. Type-SV2 pairings, on the contrary, is not being used today. A delegation protocol for pairings of this type could be considered practically useless.

3 Security Model

We formalize the security requirements of batch delegation protocols for pairings in which one of the points is a constant, which are sufficient for our proposed protocols. The requirements can however be generalized for other pairing-types.

Let $A \in \mathbb{G}_1$ be the constant point, and $B_1, \ldots, B_n \in \mathbb{G}_2$ be the variable points to be paired up with A. The protocol's goal is to compute $\{e(A, B_i) : i \in \{1, \ldots, n\}\}$. Let $\mathcal{R}$ and $\mathcal{E}$ be two probabilistic polynomial time (PPT) algorithms modeling the Delegator and the Delegatee respectively. Let $out \xleftarrow{R} \mathcal{R}(in_1)_{\mathcal{E}(in_2)}$ denotes out is the output of $\mathcal{R}(in_1)$ in the interaction with $\mathcal{E}(in_2)$, where the inputs in_1 and in_2 may or may not be equal. Finally, $\mathcal{BDH}(k)$ denotes a instance generator for groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with orders 2^k and a pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Desired properties of a batch delegation protocols for pairings include:

Completeness. The Delegator $\mathcal{R}$ obtains $\{e(A, B_i) : i \in \{1, \ldots, n\}\}$ after interacting with an honest Delegatee $\mathcal{E}$.

Secrecy. Even a cheating Delegatee $\mathcal{E}$ cannot learn any information about A and $\{B_i : i \in \{1, \ldots, n\}\}$. The simulator is given A (respectively $B_1, \ldots, B_n$) if and only if A (respectively $B_1, \ldots, B_n$) is public.

We start by giving definition for the case in which A is public. The definition for any subset of $\{B_1, \ldots, B_n\}$ being public can be defined similarly. We need to define two more notations about simulatability:

- $Trans_B(param, A, B_1, \ldots, B_n)$ denotes a PPT algorithm outputting the transcript of communication between the interaction of the Delegator $\mathcal{R}(param, A, B_1, \ldots, B_n)$ and the Delegatee $\mathcal{E}(param, A)$.

– $Sim_B(param, A)$ denotes a PPT algorithm outputting a transcript simulating the one between the interaction of the Delegator $\mathcal{R}(param,\ A,\ B_1,\ \ldots,\ B_n)$ and the Delegatee $\mathcal{E}(param, A)$.

The protocol has secrecy if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that $|\Pr[\mathbf{Game}_{\mathcal{A}}^{\mathrm{secr}_B}(k) = 1] - \frac{1}{2}|$ is negligible in k, over the random coins of all algorithms.

$\mathbf{Game}_{\mathcal{A}}^{\mathrm{secr}_B}(k)$
$param \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \xleftarrow{R} \mathcal{BDH}(k)$
$b \xleftarrow{R} \{0, 1\}$;
if $(b = 0)$
  then $\Gamma \xleftarrow{R} Trans(param,\ A,\ B_1,\ \ldots,\ B_n)$;
  else $\Gamma \xleftarrow{R} Sim(param, A)$;
$\hat{b} \xleftarrow{R} \mathcal{A}(param, A, \Gamma)$;
if $(b = \hat{b})$ then return 1 else return 0;

Correctness. The Delegator $\mathcal{R}$ can detect (with non-negligible probability) when the Delegatee $\mathcal{E}$ is cheating (i.e. the final result leads to a wrong value).

$\mathbf{Game}_{\mathcal{E}}^{\mathrm{corr}}(k)$
$param \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \xleftarrow{R} \mathcal{BDH}(k)$
$(t_1, t_2, \ldots, t_n) \xleftarrow{R} \mathcal{R}(param, A, B_1, \ldots, B_n)_{\mathcal{E}(param, A, B_1, \ldots, B_n)}$;
if $(\exists i \in \{1, \ldots, n\}, (t_i \neq e(A, B_i)))$ return 0; else return 1;

The protocol is correct if the probability that $\mathbf{Game}_{\mathcal{E}}^{\mathrm{corr}}(k) = 1$, for all $param$ generated by $\mathcal{BDH}$, for all $(A,\ B_1,\ \ldots,\ B_n) \in \mathbb{G}_1 \times {\mathbb{G}_2}^n$, and for all PPT adversary $\mathcal{E}$ (that may deviate from the protocol in arbitrary way), over the random coins of all algorithms, is non-negligible in k.

This game is basically saying that, for any cheating Delegatee $\mathcal{E}$ and for any $(A, \{B_i : i \in \{1,\ \ldots,\ n\}\})$, the Delegator $\mathcal{R}$ outputs either $\{e(A, B_i) : i \in \{1,\ \ldots,\ n\}\}$ or $\perp$, denoting invalid, except with negligible probability, even $(A, \{B_i : i \in \{1,\ \ldots,\ n\}\})$ is given to the cheating Delegatee $\mathcal{E}$.

Note that the definition of correctness in [19, 32] does not specify whether A and/or B are known to the adversary (or a malicious delegatee). Such an ambiguity could lead to insecure delegation protocols. For example, we were able to come up with a delegation protocol by simplifying one of the protocols in [32] that is secure if and only if A is private. Our adversarial model *explicitly* equips the adversary with the knowledge of all points, which is stronger than what is expected since $\mathcal{E}$ is not given some of the points in some cases.

We also remark that even both points are public, a secure pairing delegation protocol is still necessary due to the correctness requirement.

4 Our Protocols

We present our construction of four protocols for delegating the computation of pairings in batch, each for a different type of pairings. In all four protocols, the interaction is between delegator Ron and delegatee Ellen during which

Ron, given a list of n pairs of points $\langle (A, B_1), \ldots, (A, B_n) \rangle$, tries to compute $\langle e(A, B_1), \ldots, e(A, B_n) \rangle$ with the help of Ellen. The common input and private input to each of the two parties vary from one protocol to another, but the distinction should be clear from the context. At the end of the protocol runs, Ron outputs either the correct answers, or $\perp$, which indicates a failure.

In our protocols, Ron is equipped with a secret constant $e(A, Q)$ for a random and private Q. The value of such a constant may be obtained through a one-time pre-computation either by Ron himself, or by a trusted third party.

All four proposed protocols are secure under the security model defined in Section 3. The security proofs can be found in Appendix A.

4.1 Our Four Constructions

Protocol-SVSC. This protocol delegates in batch the computation of n Type-SVSC pairings as follows.

1. *(Precompute.)* Ron picks $r_A, r_Q \in_R \mathbb{Z}_p$ and computes $\tilde{A} = r_A A$, $\tilde{Q} = r_Q Q$ and $e(\tilde{A}, \tilde{Q}) = e(A, Q)^{r_A r_Q}$. This step may be precomputed before knowing the values of B_i's.
2. *(Request.)* Ron sends to Ellen $\langle \tilde{A}, \tilde{B}_0, \tilde{B}_1, \ldots, \tilde{B}_n \rangle$, where $r_i, b_i, \in_R \mathbb{Z}_p$ and $\tilde{B}_i = r_i B_i$ for $i = 1$ to n, and $\tilde{B}_0 = \tilde{Q} + \sum_{i=1}^{n} b_i \tilde{B}_i$.
3. *(Respond.)* Ellen sends to Ron $\langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle$, where $\alpha_i = e(\tilde{A}, \tilde{B}_i)$ for all $i = 0$ to n.
4. *(Verify.)* Ron verifies if $\alpha_i \in \mathbb{G}_T$ for $i = 0$ to n and $e(\tilde{A}, \tilde{Q}) \prod_{i=1}^{n} \alpha_i^{b_i} = \alpha_0$.
5. *(Output.)* Ron returns $\perp$ if the above verification failed. Otherwise he returns $\langle \alpha_1^{1/r_A r_1}, \ldots, \alpha_n^{1/r_A r_n} \rangle$.

Protocol-SVPC. To batch delegate Type-SVPC pairings, follow the steps in Protocol-SVSC, except that r_A is set to 1 instead of a random element so that $\tilde{A}$ is equivalent to A. As a result, the computation and the transmission of $\tilde{A} = r_A A$ in Step 1 and 2 are eliminated.

Protocol-PVSC. To batch delegate Type-PVSC pairings, follow the steps in Protocol-SVSC, except that all r_i's are set to 1 instead of random elements so that $\tilde{B}_i$ is equivalent to B_i for all $i = 1$ to n. The steps related to of all $\tilde{B}_i$'s of the protocol are also eliminated.

Protocol-PVPC. To batch delegate Type-PVPC pairings, follow the steps in Protocol-SVSC, except that r_A and all r_i's are set to 1 so that $\tilde{A}$ is equivalent to A and $\tilde{B}_i$ is equivalent to B_i for $i = 1$ to n. As a result, the computation of $\tilde{A}$ in Step 1, the computation of all $\tilde{B}_i$'s in Step 2, the transmission of $\tilde{A}$ in Step 2, and the exponentiation of all α_i's in Step 5 of the protocol are all eliminated.

Remarks. In the last two protocols, all the points are public and we do not

care secrecy. Nevertheless, a secure pairing delegation protocol is still necessary for the correctness requirement.

Each of our four batch protocols degenerates to a non-batch protocol if we set n to 1. We therefore obtain as a side-product four protocols for conventional pairing delegation. As will become clear in the next section, the degenerated version of our proposed protocols either performs better or is as good as the existing state-of-the-art protocols.

4.2 Performance Analysis

We now analyze the performance of the four protocols presented above in terms of both time and space complexities. Specifically, we contrast each of our four protocol constructions for delegating n pairing computations in batch to n parallel and independent invocations of traditional (non-batch) protocol for delegating pairings of the same type. We look at several aspects in our comparison, namely the size of communication, the online computational cost and total computational cost on the delegator and the total computational cost on the delegatee.

All four protocols selected for comparison are state-of-the-art in the literature in terms of efficiency. Two of them are due to Chevallier-Mames et al. [19] and the other two are due to Kang et al. [32] (see Table 1). It has been assumed in all their four protocols that the delegator already knows $e(X, Y)$ for some random X and constant Y before the protocol execution. Ron must therefore either compute $e(X, Y)$ himself or obtain the value from a trusted third party. While it is not explicitly suggested in [19, 32] which should the case, we believe that the latter is much less favorable in practice.[7] Consequently, we assume, for the sake of a fair comparison, that it is Ron himself who computes $e(X, Y)$ in these protocols, by picking a $x \in_R \mathbb{Z}_p$ and calculating $X = xP$ and $e(X, Y) = e(P, Y)^x$, where $e(P, Y)$ is a constant already known to Ron. This adds 1 $\mathbb{G}_1/\mathbb{G}_2$ Scalar Multiplication (SM) and 1 $\mathbb{G}_T$ Exponentiation (EXP) to the offline computational costs of Ron in their protocols.

The comparison is detailed in Table 1. We make a few observation here. First, the size of communication is at least halved in all four cases. Hence, our protocols give a speedup in communication of a minimum of 2. Second, Ellen sees a speed up of 2 in all four cases because she has to compute only half the number of pairings using our protocols. Also, online computational costs on Ron are roughly the same except that our Protocol-SVPC is slightly worse than that in [32, §4.3]. Finally, the total computational costs on Ron are always smaller in all our protocols. Specifically, if we ignore the computation of $\mathbb{G}_1/\mathbb{G}_2$ point additions and $\mathbb{G}_T$ multiplications (as their costs are dominated by the costs of the computation of $\mathbb{G}_1/\mathbb{G}_2$ scalar multiplication and $\mathbb{G}_T$ exponentiation respectively), the speed up ranges from 1.3 to 4, depending on the relative speed of computing SMs and EXPs.

[7] The following must hold for the protocols to remain secure: the value of $e(X, Y)$ is correct, $e(X, Y)$ is known only to Ron, and Y is known only to Ron in cases where Y represents some secret of Ron's. These imply a trusted third party with high trustworthiness is required, the maintenance of which is costly.

Table 1. Performance comparison between our proposed protocols and multiple runs of state-of-the-art non-batch protocols when the batch size is n

Type	Protocol	Communication Complexities		Computation at Ron (#SMs,#PAs,#EXPs,#MULs)*		Computation at Ellen
		#rounds	(#$\mathbb{G}_1$/$\mathbb{G}_2$, #$\mathbb{G}_T$)	Online cost	Total cost	(#Pairings)
SVSC	[32, §4.2]	2	(4n,2n)	(n,n,4n,n)	(5n,n,5n,n)	2n
	Our Protocol-SVSC	**2**	**(n+2,n+1)**	**(2n,n,3n,n)**	**(2n+2,n,3n+1,n)**	**n+1**
SVPC	[32, §4.3]	2	(2n,2n)	(n,n,3n,n)	(3n,n,4n,n)	2n
	Our Protocol-SVPC	**2**	**(n+1,n+1)**	**(2n,n,3n,n)**	**(2n+1,n,3n+1,n)**	**n+1**
PVSC	[19, §6.2]	2	(4n,2n)	(n,n,3n,n)	(4n,n,4n,n)	2n
	Our Protocol-PVSC	**2**	**(n+2,n+1)**	**(n,n,3n,n)**	**(n+2,n,3n+1,n)**	**n+1**
PVPC	[19, §6.1]	2	(2n,2n)	(n,n,2n,n)	(2n,n,3n,n)	2n
	Our Protocol-PVPC	**2**	**(n+1,n+1)**	**(n,n,2n,n)**	**(n+1,n,2n+1,n)**	**n+1**

* PA and MUL denote $\mathbb{G}_1$/$\mathbb{G}_2$ point addition and $\mathbb{G}_T$ multiplication respectively.

4.3 Implementation and Experimentation

The fact that computationally limited devices such as smartcards are slow and resource-constrained for computing pairings has been a major motivation for delegation in [19, 32]. By delegating the computation to more powerful entities such as PCs on which efficient libraries for pairing computation exist, it becomes practical for those devices to execute pairing-based cryptographic algorithms.

In this regard, pairing delegation makes sense only if it is faster for an entity to delegate the computation than doing the work itself. For instance, Scott et al. suggested recently in [37] that delegation might not offer any benefit because it does not offer significant gain in speed. In particular, the time for computing a pairing is comparable to the time for doing other underlying group operations.[8]

Nevertheless, one should not project from such a result, that was obtained from one experimental testbed with a particular set of parameter choices, and conclude that pairing delegation does not make sense in all scenarios. In fact, the relative speed of computing pairings and various underlying group operations can vary greatly depending on numerous factors ranging from the choices of curves and fields and their representation, to instruction sets, compilers and libraries in case of general-purpose processors, or architecture and level of parallelism in case of dedicated hardware coprocessor implementation. Pairing delegation speeds up pairing computation at the protocol level independent of the choice of any parameters mentioned above. We believe that the pairing delegation offers significant benefits in many real-world settings.

To gain some empirical data, we implemented our four proposed protocols in `C` using the PBC library[9] (version 0.4.7) for its elliptic-curve and pairing operations. The machine we used for the experiments was a Lenovo T60 laptop PC with an Intel dual-core 2GHz CPU and 1.5GB of Ram, running Ubuntu 6.10. Timing figures collected using such a platform serve as a heuristic of the

[8] This is indeed an exciting result because it demonstrates researchers' success in speeding up pairing computation and hence making PBC ever-increasingly practical.

[9] http://crypto.stanford.edu/pbc/

lower bound on the efficiency gain using our batch delegation protocols because pairing computation has already been made very efficient on such a platform.

We did experiments to compare the latency of computing pairings locally (without any delegation) to that of using our four protocols, over various types of underlying elliptic curves.[10] In particular, we measured the sum of the time our protocols spent on performing the *Request* and the *Verify* steps. This measurement reflects the latency experienced by the delegator if we ignore the communication overhead and assume that the delegatee can compute pairings in relatively no noticeable time. Experiments were repeated 10 times using random input points over which timings were averaged. We used a batch size of 100.[11]

We calculated the speedups gained from using our protocols over computing pairings locally. For example, using a Type-F curve, it took 7.87s to compute the 100 pairings locally while a latency of 3.34s was measured when our Protocol-SVSC was used. The speedup was thus 7.87s/3.34s=2.34. Empirical results for other experimental parameters are summarized in Table 2. As shown by the existence of speedups that are less than 1, delegation did not always outperform local computation in terms of speed. This happens, for example, for curves over which computing a pairing and computing an SM and/or an EXP take comparable time. However, delegation did shorten the latency of pairing computation in the majority of cases and some of the speedups were significant.

Table 2. The speedup gained by using our proposed protocols over computing the 100 pairings locally using different types of curves as defined in the PBC Library

Curve	**Protocol-SVSC**	**Protocol-SVPC**	**Protocol-PVSC**	**Protocol-PVPC**
Type-A	0.75	**1.75**	0.73	**2.20**
Type-A1	**1.13**	**2.59**	**1.13**	**3.23**
Type-D225	0.49	**1.01**	0.49	**1.48**
Type-E	**1.17**	**3.22**	**1.20**	**3.60**
Type-F	**2.34**	**2.67**	**2.36**	**6.73**
Type-G	0.69	**1.29**	0.67	**2.04**

5 An Application Scenario

In this section, we propose a new application of pairing delegation and discuss some less apparent costs of delegation.

A Novel Application in Trusted Computing. An application for pairing delegation that has not been considered so far falls into the arena of trusted computing. Research in trusted computing strives to raise the trustworthiness of computing devices such as PCs by guaranteeing that they operate correctly and securely even under certain software and/or hardware attacks. To achieve this goal,

[10] Please refer to the PBC Library for the characteristics of these curves.

[11] As shown earlier, the batch size has little effect on the timing of the delegator.

trusted computing almost always uses some kind of tamper-resistant, tamper-evident, and/or tamper-responsive hardware. For example, the IBM 4758/4764 secure coprocessor [39] is a general-purpose PC built into a PCI card that is wrapped around in an armor made of layers of material capable of detecting tampering attempts. *Trusted Computing Group (TCG)*'s *Trusted Platform Module (TPM)*[12] takes a radically different approach—TPMs provide minimal tamper-resistant storage barely enough for storing a handful of cryptographic key material, leaving most of the operation outside the physical protection boundary. Being cheap enough to be deployed in all PCs was one of the design goals.

The Challenge. In general, we are faced with the following dilemma when building any trustworthy device, be it a general-purpose machine that runs arbitrary software, or a dedicated chip devised for a specific application—On one hand, we would like to put as much circuitry as possible into the *Trusted Computing Base (TCB)* to make the device more powerful. But on the other, it gets very pricey as the protection boundary grows due to increased complexity in configuration, maintenance and upgrade. Heat dissipation also places a practical limit on how powerful the hardware inside the TCB could possibly get.

A specific instance of the above challenge we would like to solve in this paper is *how to build a cost-effective pairing coprocessor with high assurance in physical security.* We would like to compute pairings correctly and securely despite the potential presence of various software and/or hardware attacks, because insecure pairing computation implies whatever cryptographic protocols for which these pairings are computed become insecure.

Our Solution. Pairing delegation provides a solution to the problem. One could build a secure coprocessor for pairing computation using two co-operating modules that are unbalanced in their size, power and trustworthiness. Specifically, one of the modules is lightweight but physically secured, which plays the role of the pairing delegator. The other is powerful but physically unprotected, and acts as the pairing delegatee, as shown in Figure 1. The security of pairing delegation guarantees that a pairing coprocessor built in this way operates correctly and securely even if only the small delegator module is physically armored. Consequently, such a coprocessor is lower in cost thanks to a smaller TCB, but still achieves good performance because the actual heavyweight pairing computation is now performed by the more powerful delegatee module.

Building a pairing coprocessor using the delegation approach as described above also allows more fine-grained engineering decisions for higher flexibility and better optimization. It is possible, for instance, that the delegator uses a multiplier module of smaller digital size for the underlying field arithmetics owing to limitation on space and/or cost while the delegatee does field multiplication in software on a multi-core PC for lower non-recurring engineering costs. This approach also permits upgrade, maintenance and reconfiguration of the delegatee accelerators without having to re-engineer the TCB.

[12] https://www.trustedcomputinggroup.org/home

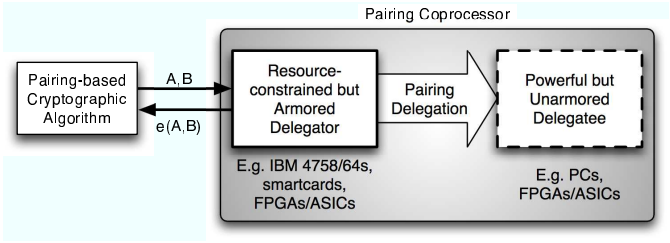

Fig. 1. An architecture for a cost-effective and trustworthy secure coprocessor for pairing computation that leverages pairing delegation

Some Less Apparent Costs. Pairing delegation incurs additional costs that must be taken into account so as to correctly decide whether it is more cost-effective to delegate the computation of pairings. Below we mention some of the less apparent costs that might get ignored.

Denial-of-Service (DoS) Attacks. Delegation protocols are inherently vulnerable to DoS attacks as the delegator needs cooperative help from the delegatee. An adversary can launch DoS attacks by corrupting the delegatee, or simply jamming the channel. Batch delegation protocols are even more vulnerable as corrupting the value of one pairing in the batch renders all the pairings useless.[13]

Timing-analysis Attacks. Cryptographic protocols are in general more susceptible to timing attacks than cryptographic algorithms due to the existence of communication events, the timing of which may be correlated to some internal secret values. Additional measures must be taken to compensate this extra attack surface opened up due to the delegation protocols' necessity to communicate.

Source of Randomness. Pairing delegation requires a cryptographically secure *Random Number Generators (RNG)*, while computing pairings locally does not. A secure RNG relies on special hardware to collect random physical events and incurs costs that vary depending on platform architectures.

Communication Costs. Communication could dominate the whole protocol execution time in the case when the communication channel has a low bandwidth, and/or a high latency. Delegation would thus be undesirable. Also, energy-constrained devices such as sensor nodes may prefer not to delegate because communication is very costly in terms of power consumption.

Latencies. Gains in efficiency by delegating pairings in batch grow with the size of the batch. However, computation latencies also grow with batch sizes, as the answer to the computation of any pairing in the batch won't be available until the delegatee has completed computing the whole batch. There is thus a tradeoff between efficiency and latencies. For instance, while batch protocols are useful

[13] A potential solution would be to employ some error-detection techniques to identify the "bad" pairings in a batch.

in case of verifying aggregate signatures, they might be unsuitable if pairing computation occurs rarely, or answers to the computation are needed promptly.

6 Conclusions

In this paper, we have introduced the concept of batch pairing delegation and provided four protocol constructions, each for a different pairing-type. The security of our protocols has been proven under a new security model we formalized. We have implemented our protocols in software for experimentation, and shown that our batch delegation protocols are more efficient than both individually delegating pairing computation using the state-of-the-art non-batch protocols and computing the pairings locally without any delegation. Moreover, we have proposed how to build a secure coprocessor for pairing computation cost-effectively using delegation, and discussed some of the less apparent costs of delegation.

References

1. M. Abdalla, E. Kiltz, and G. Neven. Generalized key delegation for hierarchical identity-based encryption. In *ESORICS*, LNCS. Springer, 2007. To appear.
2. J. Baek and Y. Zheng. Identity-based threshold decryption. In *Public Key Cryptography*, LNCS 2947, pages 262–276. Springer, 2004.
3. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT*, pages 236–250, 1998.
4. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
5. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-Group signature scheme. In *Public Key Cryptography*, LNCS 2567, pages 31–46. Springer, 2003.
6. D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT*, LNCS 3027, pages 223–238, 2004.
7. D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 2008. To Appear.
8. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, LNCS 2139, pages 213–229. Springer, 2001.
9. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, LNCS 2656, pages 416–432. Springer, 2003.
10. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference (TCC)*, LNCS 3378, pages 325–341, 2005.
11. D. Boneh and B. Waters. A collusion resistant broadcast, trace and revoke system. In *ACM Conference on Computer and Communications Security (CCS)*, 2006.
12. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554. Springer, 2007.
13. X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *Public Key Cryptography*, LNCS 4450, pages 1–15. Springer, 2007.
14. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 2007. To Appear.

15. J. C. Cha and J. H. Cheon. An identity-based signature from Gap Diffie-Hellman groups. In *Public Key Cryptography*, LNCS 2567, pages 18–30. Springer, 2003.
16. M. Chase and A. Lysyanskaya. Simulatable VRFs with applications to multi-theorem NIZK. In *CRYPTO*, LNCS. Springer, 2007. To Appear.
17. L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241, 2007.
18. L. Chen and C. Kudla. Identity based authenticated key agreement protocols from pairings. In *IEEE Computer Security Foundations Workshop*, pages 219–233, 2003.
19. B. Chevallier-Mames, J.-S. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive, 2005/150.
20. S. S. M. Chow. Verifiable pairing and its applications. In *Information Security Applications (WISA)*, LNCS 3325, pages 170–187. Springer, 2004.
21. S. S. M. Chow, C. Boyd, and J. M. G. Nieto. Security-mediated certificateless cryptography. In *Public Key Cryptography*, LNCS 3958, pages 508–524, 2006.
22. S. S. M. Chow and K.-K. R. Choo. Strongly-secure identity-based key agreement and anonymous extension. In *Information Security (ISC)*, LNCS, 2007. To appear.
23. S. S. M. Chow, S.-M. Yiu, L. C. K. Hui, and K. P. Chow. Efficient forward and provably secure ID-based signcryption scheme with public verifiability and public ciphertext authenticity. In *ICISC*, LNCS 2971, pages 352–369. Springer, 2003.
24. A. W. Dent. A survey of certificateless encryption schemes and security models. Cryptology ePrint Archive, Report 2006/211, 2006.
25. Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, LNCS 3386, pages 416–431. Springer, 2005.
26. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006. http://eprint.iacr.org/.
27. C. Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, LNCS 4004, pages 445–464. Springer, 2006.
28. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.
29. J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, LNCS 4004, pages 339–358. Springer, 2006.
30. F. Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, LNCS 2595, pages 310–324. Springer, 2002.
31. A. Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS*, LNCS 1838, pages 385–394. Springer, 2000.
32. B. G. Kang, M. S. Lee, and J. H. Park. Efficient delegation of pairing computation. Cryptology ePrint Archive, Report 2005/259, 2005. http://eprint.iacr.org/.
33. K. Kurosawa and S.-H. Heng. The power of identification schemes. In *Public Key Cryptography*, LNCS 3958, pages 364–377, 2006.
34. J. Pastuszak, D. Michatek, J. Pieprzyk, and J. Seberry. Identification of bad signatures in batches. In *Public Key Cryptography*, LNCS 1751, pages 28–45, 2000.
35. J. Pastuszak, J. Pieprzyk, and J. Seberry. Codes identifying bad signature in batches. In *INDOCRYPT*, volume 1977 of *LNCS*, pages 143–154. Springer, 2000.
36. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.
37. M. Scott, N. Costigan, and W. Abdulwahab. Implementing cryptographic pairings on smartcards. In *CHES*, volume 4249 of *LNCS*, pages 134–147. Springer, 2006.
38. E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

39. S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(8):831–860, 1999.
40. D. Yao, N. Fazio, Y. Dodis, and A. Lysyanskaya. ID-based encryption for complex hierarchies with applications to forward security and broadcast encryption. In *ACM CCS*, pages 354–363, 2004.

A Security Proofs

Completeness is trivial. Secrecy is also easy to see. The random choice of r_A and/or the random choices of $r_1, \cdots, r_n$ (depends on which of A and B's are public) each time makes the view of the delegatee follows the same distribution.

The following proves that all of our four proposed delegation protocols satisfy the requirement of correctness. *A single proof* is sufficient since its crux depends on the randomness of r_Q that is present in all our proposed protocols.

Note that we give an *information theoretic* argument here. Moreover, all our protocols are single-round, so they are secure against "concurrent attacker".

Since $e(P, Q)$ is a generator of $\mathbb{G}_T$, we assume without loss of generality that the adversary returns the values of $\{\alpha'_i : i \in \{0, 1, \cdots, n\}\}$ in the following forms:

$$\alpha'_i = e(\tilde{A}, \tilde{B}_i)e(P, Q)^{\beta_i}, \forall i \in \{0, 1, \cdots, n\}, \text{ where } \beta_i \in \mathbb{Z}_p, \forall i \in \{0, 1, \cdots, n\}$$

To break the correctness, $\exists j \in \{1, \cdots, n\}$ such that $\beta_j \neq 0$. Note that any other β_is, in particular β_0, can take any value arbitrarily.

Suppose $\alpha_i = e(\tilde{A}, \tilde{B}_i), \forall i \in \{0, 1, \cdots, n\}$, which means $\alpha_0 = e(\tilde{A}, \tilde{Q}) \prod_{i=1}^{n} \alpha_i^{b_i}$. For verification not to return $\perp$, we have the following equation holds:

$$\begin{aligned}
\alpha'_0 &= e(\tilde{A}, \tilde{Q}) \prod_{i=1}^{n} {\alpha'}_i^{b_i} \\
\alpha_0 \cdot e(P, Q)^{\beta_0} &= e(\tilde{A}, \tilde{Q}) \prod_{i=1}^{n} (\alpha_i \cdot e(P, Q)^{\beta_i})^{b_i} \\
e(P, Q)^{\beta_0} &= \prod_{i=1}^{n} e(P, Q)^{\beta_i \cdot b_i} \\
\beta_0 &= \sum_{b=1}^{n} b_i \beta_i \\
b_j \beta_j &= \beta_0 - \sum_{i \in \{1, \cdots, n\} \setminus \{j\}} b_i \beta_i
\end{aligned}$$

From $\tilde{B}_0 = r_Q Q + \sum_{i=1}^{n} b_i \tilde{B}_i$, as long as r_Q is randomly chosen each time, there are always p possibilities for b_j. Without loss of generality, we assume $\alpha'_j \neq e(A, B_j)^{1/r_A r_j}$, i.e. $\beta_j \neq 0$. With b_j being an unknown, the probability that the above equality holds is at most $1/p$. Note that we made no assumption on other α'_is for $i \neq j$ are correct or $(A, \{B_i : i \in \{1, \cdots, n\}\})$ being unknown to the cheating delegatee. However, we do assume Q is kept secret from the delegatee, otherwise partial information about r_Q will be leaked when the delegatee knows A and/or $\{B_i : i \in \{1, \cdots, n\}\}$.