Towards a Practical Cryptographic Voting Scheme Based on Malleable Proofs

Full version — the original appears at Vote-ID 2013

David Bernhard[1], Stephan Neumann[2], and Melanie Volkamer[2]

[1] University of Bristol, United Kingdom
[2] CASED / TU Darmstadt, Germany

Abstract. Mixnets are one of the main approaches to deploy secret and verifiable electronic elections. General-purpose verifiable mixnets however suffer from the drawback that the amount of data to be verified by observers increases linearly with the number of involved mix nodes, the number of decryptors, and the number of voters. Chase et al. proposed a verifiable mixnet at Eurocrypt 2012 based on so-called *malleable proofs* - proofs that do not increase with the number of mix nodes. In work published at PKC 2013, the same authors adapted malleable proofs to verifiable distributed decryption, resulting in a cryptographic voting scheme. As a result, the amount of data to be verified only increases linearly with the number of voters. However, their scheme leaves several questions open which we address in this paper: As a first contribution, we adapt a multi-party computation protocol to build a distributed key generation protocol for the encryption scheme underlying their voting scheme. As a second contribution, we decompress their abstract scheme description, identify elementary operations, and count the number of such operations required for mixing and verification. Based on timings for elementary operations, we extrapolate the running times of the mixing and verification processes, allowing us to assess the feasibility of their scheme. For the German case, we conclude that the replacement of postal voting by cryptographic voting based on malleable proofs is feasible on an electoral district level.

Keywords: Malleable Proofs, Distributed Key Generation, Performance

1 Introduction

Since Chaum's seminal work [1], many cryptographic voting schemes have been proposed aiming for secret and verifiable elections. Beside blind signatures and homomorphic tallying, the use of mixnets has gained lots of interest in the research community. The success of the mix-based approach is largely due to recent mathematical achievements with respect to verifiable mixnets for large-scale elections, e.g., Wikström [2–4], Lipmaa and Zhang [5], and Bayer and Groth [6]. In the mix-based approach, the election is usually conducted in the following way: Voters individually encrypt their votes with the public key of the election authority and publish the resulting ciphertexts on a bulletin board. After the declared voting phase, a mixnet is used to anonymize all encrypted votes from eligible voters such that after the anonymization process, individual, encrypted votes can be decrypted by the election authority.

Mixnets are instantiated by independent mix nodes. Each mix node in turn verifies the proofs of all predecessor mix nodes, re-randomises and shuffles the encrypted votes, and adds a non-interactive zero-knowledge proof to its output attesting that it has shuffled correctly. The election authority is instantiated by a set of decryptors (often referred to as trustees). After the anonymization process, each decryptor partially decrypts the list of anonymized ciphertexts and generates a proof that it has partially decrypted this list correctly. Individual plaintexts can be reconstructed by combining a threshold number of partial decryptions. These plaintext votes are used afterwards to calculate the election result. After the tallying, observers (who might be individual voters) verify all proofs generated by all mix nodes and decryptors to convince themselves that the announced election result is correct. The amount of data to be processed by each observer depends linearly on the number of mix nodes, the number of decryptors, and the number of voters.

At Eurocrypt 2012, Chase et al. invented the concept of *malleable proof systems* [7]. Rather than generating individual and independent proofs, malleable proofs allow an individual mix node $i+1$ to "update" the zero-knowledge proof π_i of mix node i and to add another permutation and randomisation, resulting in proof π_{i+1}. The updated proof is of the same general form as the original one, only the constants having changed. Therefore, the amount of data to be verified only increases linearly with the number of decryptors and voters but is independent of the number of mixers. Chase et al. propose using the DLIN encryption scheme [8] and Groth-Sahai proofs [9] in the DLIN setting to instantiate their construction. The appeal of malleable proofs has motivated work published recently at PKC 2013 [10]. In this work, the authors adapt malleable proofs to distributed decryption and thereby instantiate a cryptographic voting scheme (henceforth referred to as the CKLM13 scheme), which forms the basis of the current work. In CKLM13, the amount of data to be processed by each observer only increases linearly with the number of voters.

Our Contribution. While the underlying ideas and constructions are of great theoretical value, so far the practical use of malleable proofs within cryptographic voting schemes was beyond the scope of the work by Chase et al. [10]. Specifically, two practical questions remain open, which are addressed in our paper. 1) Though proposing a cryptographic voting scheme based on malleable proofs, to date there is no distributed key generation protocol for DLIN known and therefore CKLM13 implicitly relies on a single trusted key distribution party. 2) The concept and the instantiation of malleable proofs for cryptographic voting have been highly theoretical and an evaluation of the real-world feasibility of tallying and verification in terms of computational efficiency is pending.

To address the first problem, we propose a distributed key generation protocol for the DLIN encryption scheme. This allows us to extend the CKLM13 scheme to a fully distributed cryptographic voting scheme. We do so by adapting a multi-party computation (MPC) protocol invented by Smart and Geisler [11] to the DLIN encryption scheme. The distributed key generation protocol comes at the cost of the assumption that at most $t < n/3$ election administrators (decryptors) are actively cheating. While we concede that cryptographic elections should be verifiable even if all administrators are dishonest, we point out that this problem has not previously been addressed at all for the DLIN encryption scheme: to the best of our knowledge, no DLIN key generation algorithm has been proposed to date that is secure against even *one* dishonest participant.

In the remainder of this work, we refer to this extended version of CKLM13 as our *modified CKLM13* scheme. To answer the second question, we investigate the CKLM13 scheme in detail and expand its abstract description. This allows us to identify and count elementary operations. Using timings from the MIRACL pairing-based cryptography library [12], we draw conclusions about the real-world feasibility of cryptographic voting schemes based on malleable proofs. With reference to the election statistics of Darmstadt, Germany, we conclude that the postal voting process can be replaced by cryptographic voting based on malleable proofs, while on a city level the application of cryptographic voting based on malleable proofs is beyond practical use.

Structure. The remainder of this work is structured as follows: In Section 2, we provide the reader with preliminaries used throughout the paper. Section 3 is dedicated to the construction of a distributed key generation protocol for the DLIN encryption scheme. Thereafter, in Section 4, we analyze the Groth-Sahai proofs used in CKLM13 with respect to elementary operations. Based on implementation timings of the underlying cryptography, we draw conclusions about the feasibility of the modified scheme. Finally, we conclude our paper and give directions for future work in Section 5.

2 Preliminaries

In this section we introduce the notation and cryptographic concepts that we use in our work.

Notation. We denote assignment of value a to variable x by $x \leftarrow a$; assigning to x a value chosen uniformly at random from set S we denote by $x \twoheadleftarrow S$. In cryptographic groups we denote group elements by capital letters and integers (modulo the group order) by small ones. Algorithm names are set in $\mathsf{SansSerif}$.

Public-Key Threshold Encryption. A public-key encryption scheme is a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ where KeyGen takes a security parameter as input and produces a public and a secret key; $\mathsf{Encrypt}$ takes a message and a public key and produces a ciphertext and $\mathsf{Decrypt}$ is deterministic, takes a secret key and a ciphertext as input and returns a message.

A threshold encryption scheme is characterised by two parameters, a number of decryptors n and a security threshold $t < n$. Informally, the properties we want are that any subset of at least $t+1$ decryptors can jointly decrypt ciphertexts but no set of size at most t can gain any information from ciphertexts. In particular, at no point in a setup–encryption–decryption cycle is any one party or subset of size at most t in possession of a full decryption key. Following Fouque, Pointcheval and Stern [13], a threshold key generation scheme is defined by a 4-tuple of algorithms[3]: KeyGen takes a security parameter as input and outputs a public key pk and n key shares sk_i for the decryptors; $\mathsf{Encrypt}$ takes a message m and a public key pk and outputs a ciphertext c; $\mathsf{Decrypt}$ takes a ciphertext c and a key share sk_i and outputs a decryption share d_i; $\mathsf{Combine}$ takes a ciphertext c and a set of at least $t+1$ decryption shares d_i and outputs either a message m or the special symbol $\perp$ to denote failure. For any public key pk and set of key shares $(sk_i)_{i=1}^n$ produced by KeyGen, for any message m and any ciphertext c produced by $\mathsf{Encrypt}$ on m and pk and for any subset $S \subseteq \{1, \ldots, n\}$ of size $|S| = t+1$ it must hold that if for all $s \in S$ we compute $d_s \leftarrow \mathsf{Decrypt}(c, x_s)$ then $\mathsf{Combine}(c, (d_s)_{s \in S})$ returns m. This property is known as *correctness.*

DLIN Encryption. As opposed to many other cryptographic voting schemes, CKLM13 builds upon the DLIN (also known as BBS after its authors [8]) encryption scheme, which relies on the weaker *decisional linear Diffie-Hellman* (DLIN) assumption rather than the *decisional Diffie-Hellman* (DDH) assumption to achieve IND-CPA security. The scheme lives in a cyclic group $\mathbb{G}$ with generator G of some order q, a prime power.

A secret key is a pair $(x, y) \twoheadleftarrow \mathbb{Z}_q \times \mathbb{Z}_q$ and the corresponding public key is $(X, Y) = (G^x, G^y)$. To encrypt a $M \in \mathbb{G}$ one picks a pair $(r, s) \twoheadleftarrow \mathbb{Z}_q \times \mathbb{Z}_q$ and computes $(A, B, C) \leftarrow (X^r, Y^s, MG^{r+s})$. To decrypt one recomputes M as $C/(A^{1/x}B^{1/y})$ where the inversions are taken over the field $\mathbb{F}_q$. DLIN encryption, like ElGamal, is homomorphic: the componentwise group operation on two ciphertexts is a ciphertext for the group operation on the two underlying messages. This property allows a ciphertext to be re-randomised by adding an encryption of the neutral element in $\mathbb{G}$ which forms the basis for the use of DLIN encryption in mixnets. Creating a threshold version of DLIN encryption is the subject of Section 3.

Pairing-Based Cryptography. A *pairing group* is a triple of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of some order q with an efficiently computable bilinear, non-degenerate map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ i.e. for generators G_1, G_2 of $\mathbb{G}_1, \mathbb{G}_2$ respectively and integers a, b we have $e(aG_1, bG_2) = e(G_1, G_2)^{ab}$ and $e(G_1, G_2)$ is again a generator of $\mathbb{G}_T$.

The only known implementations of such groups that are useful for cryptography are based on elliptic curves; such an implementation is called *symmetric* if $\mathbb{G}_1 = \mathbb{G}_2$ and *asymmetric* if the two groups are different and no efficient homomorphisms are known between them.

Shamir's Secret Sharing Scheme. Shamir's secret sharing scheme [14] allows a party to share a secret among any n parties such that any subset of $t \leq n$

[3] The original definition also contains verification keys, which we view as part of the public key, and mentions decryption proofs explicitly which we view as part of the decryption shares.

parties can reconstruct the secret but any smaller subset gains no information about the secret. Each party obtains as a share the value of a degree-t bounded polynomial at a distinct index over a suitable finite field such that the secret is the value at some other index (usually 0). Given any t shares, the secret can be reconstructed by interpolation using Lagrange coefficients.

3 Key Generation with Multi-Party Computation

Voting is among the most security-sensitive applications that can be conducted over the Internet. Therefore, complex trust distribution concepts are in place to prevent malicious collaborations among internal/external attackers from violating the desired security properties. As opposed to ElGamal, which is often considered the standard in the cryptographic voting community, the DLIN encryption scheme does not come with a distributed key generation protocol. Hence, the CKLM13 scheme [10] implicitly relies on a trusted key distribution party, which forms a crucial security bottleneck of the overall scheme. This section is organised as follows: First, we explain the distinction between a key generation algorithm and a protocol and show why we want the latter, but for DLIN this does not follow directly from the former. Next, we give the key generation algorithm for DLIN, introduce multi-party computation and deploy it to turn the algorithm into a protocol. We analyse our new protocol with respect to efficiency and security. Finally, we show how our protocol greatly simplifies the threshold decryption operation.

3.1 Security of Threshold Encryption: Algorithm versus Protocol

The definition of threshold encryption (for example, [13]) only postulates a key generation *algorithm* which gives security if it is run by a trusted party who then securely distributes the key shares to the decryptors. In practice, what is required however is a key generation *protocol* that the decryptors can run jointly and that, following our informal specification, never puts any one party in possession of a key with which it could decrypt messages directly [16].

For the ElGamal encryption scheme, constructing a threshold key generation protocol is comparatively easy although not without subtleties [17]. This may be a reason that the distinction between threshold key generation algorithms and protocols is usually not made in the literature. For DLIN however, it is not an easy task to construct a secure key generation protocol from the algorithm, without a trusted party.

DLIN encryption uses two public keys X and Y. Since their use is completely symmetric, we discuss the problem relating only to a single public key $X = G^x$ for some secret x. During decryption, one raises a ciphertext component A to the power $1/x$. The threshold key generation *algorithm* therefore picks an x, creates a public key G^x, computes $\bar{x} \leftarrow 1/x$ and creates shares $\bar{x}_i$ of $\bar{x}$ — the decryptors get shares of the *inverse* of the element used as the exponent of the public key. One could try and build a protocol that starts with all parties generating shares of x and interpolating G^x. However, the shares of the inverse $1/x$ are not the same as the inverses of shares of x and there is no easy method to obtain one from the other. One might think that one could simply start with shares of $\bar{x} = 1/x$ instead but then one cannot easily compute the public key which is now $G^{1/\bar{x}}$.

3.2 A Threshold Algorithm

To construct a threshold scheme for DLIN encryption, we start with Shamir's secret sharing scheme. This gives a key generation algorithm but not yet a protocol: pick a DLIN key pair and Shamir-share the decryption keys. Shamir's scheme has homomorphic properties that allow shares to be used for decryption without ever reconstructing the key. In more detail, if $\mathbb{G}$ is a cyclic group of order q with generator G and $(x_i)_i$ are Shamir-shares of a secret x in $\mathbb{F}_q$ then $(G^{x_i})_i$ are Shamir-shares of G^x since $\mathbb{G}$ can be viewed as a vector space over $\mathbb{F}_q$ and the polynomial p defined by the shares can be lifted from $\mathbb{F}_q$ to G. This leads to the following threshold DLIN scheme.

KeyGen *(algorithm.)* For given t and n, pick secret keys $x, y \twoheadleftarrow \mathbb{F}_q$. Compute $\bar{x} \leftarrow 1/x$ and $\bar{y} \leftarrow 1/y$ over $\mathbb{F}_q$. Create a (t, n) Shamir-sharing of $\bar{x}$ and give each decryptor her share $\bar{x}_i$; repeat for $\bar{y}$. Output the public key $(X, Y) \leftarrow (G^x, G^y)$.

Encrypt Like for non-threshold DLIN encryption.

Decrypt(A, B, C) For the decryptor with shares $\bar{x}_i, \bar{y}_i$, create a decryption share as $D_i \leftarrow A^{\bar{x}_i} B^{\bar{y}_i}$.

Combine Given any set of at least $t + 1$ decryption shares $(D_s)_{s \in S, |S| > t}$, interpolate D as the value at 0 of the degree-t-bounded polynomial p such that $p(s) = D_s$ for all $s \in S$. The final decryption is $M \leftarrow C/D$.

This scheme still leaves open two questions. The first is how to check if the values D_s provided by the decryptors are correct - Chase et al. [10] suggest using malleable proofs here. Our solution to the second problem will remove the need for such proofs completely. The second problem to which Chase et al. do not provide a solution is, as mentioned, how to turn the key generation algorithm above into a protocol and eliminate the trusted party that generates keys.

3.3 Multi-Party Computation

Multi-Party Computation (MPC) [18] is the theory of cryptographic protocols in which a set of parties $\{P_i\}$, each holding some secret input x_i, jointly compute some function $(y_i)_i = f((x_i)_i)$ of their inputs in a manner as secure as if everyone sent their x_i to a trusted party who computed f and returned the appropriate y_i to each P_i.

Our starting point is the MPC protocol by Smart and Geisler [11] for identity-based schemes that require exponent inversion. The key technique in this protocol is a development of an idea by Bar-Ilan and Beaver [19] for group element inversion. Smart and Geisler's protocol was originally given for distributed decryption in a class of identity-based encryption schemes making use of exponent inversion; another protocol by Kate and Goldberg [20] achieves distributed key generation for this class of schemes but their techniques do not translate into our scenario: in IBE, one party ends up in possession of a decryption key whereas in an election, no-one should ever be able to decrypt individual ballots.

We make two minor modifications to the Smart-Geisler protocol. First, we adapt the protocol to DLIN encryption. Secondly, the original protocol runs a fast key generation followed by comparatively costly MPC for decryption whereas we are considering a scenario in which decryption operations are much more time-critical than key generation so we prefer to use MPC to generate keys and a faster decryption operation. (Technically, Smart and Geisler propose generating shares of x where the public key is $X = G^x$ and using MPC to raise an element to $1/x$ at decryption time; we generate shares of $\bar{x} = 1/x$ for decryption and use MPC to compute the public key as $G^{1/\bar{x}}$.)

Security threshold. Our protocol, as an artifact of the MPC protocol that we use, requires a security threshold $t < n/3$. We assume that broadcasting a value to all parties and sending a value privately to another party are possible. Our protocol is then secure against up to t parties actively cheating i.e. sending false values during the protocol. Constructing such a protocol for larger t we leave as an open problem. In return for this restriction on the size of t, we obtain not only the first threshold DLIN key generation *protocol* but also a very efficient one that allows us to dispense with the zero-knowledge proofs that threshold schemes usually require at decryption time.

Secure Interpolation. When reconstructing a degree-t Shamir-shared secret, as long as at least $t + 1$ of the shares are correct the presence of any incorrect shares can be detected if the following secure interpolation procedure is used. Pick a set $S \subseteq \{1, \ldots, n\}$ of any $t + 1$ indices and interpolate the secret from these (as the value at 0 of the degree-t bounded polynomial going through (i, x_i) for all $i \in S$, where x_i is the share at index i). Next, again using the set S interpolate the values of the polynomial at the indices of all other shares outside S and check that these points correspond to the actual shares received. If any of these checks fail, abort the setup protocol.

3.4 Our Protocol

Our MPC protocol will let each decryptor obtain a decryption key share $\bar{x}_i$ and all decryptors will obtain a public key $X = G^x$ where $\bar{x} = 1/x$. (To obtain the shares for the other public key component Y too, one runs the protocol twice in parallel.) We give the protocol from the point of view of a party P_i. In our protocol, we denote by $x^{(j)}$ a value received from party j, either as a broadcast or a private message; for $j = i$ this refers to the local variable x of party P_i (this notation allows us to iterate or sum over indices). We denote values that are common to all players with a star, i.e. u^*.

PRSS. We begin by setting up a Pseudo-Random Secret Sharing (PRSS) [15]. For details see Appendix B.2. This allows all parties to repeatedly draw values

x_l for any label l that form a sharing of some fresh random secret x_l^*. This need only be done once for each set of decryptors; they can generate many sets of keys with the same PRSS by picking fresh labels each time.

Decryption key shares. Once the PRSS is set up, draw a value $\bar{x}$ that will be your decryption key share. Also draw a further value r. (W.l.o.g. the labels for $\bar{x}$ and r are known to all parties.) Compute[4] $u \leftarrow \bar{x} \cdot r$. Since $\bar{x}$ and r were both degree-t shares of their respective secrets, the shares u are now degree-$2t$ shares of a value u^*.

Round 1: sharing u. Locally create a degree-t sharing of u and send each P_j her share u_j. This can be achieved by letting $c_0 \leftarrow u$ and picking random coefficients $c_1, \ldots, c_t \twoheadleftarrow \mathbb{Z}_q$ to define a polynomial $p(x) = \sum_{k=0}^{t} c_k x^k$ and letting $u_j \leftarrow p(j)$ for all j.

Round 2: interpolating u'. Collect shares $u^{(j)}$ from all other parties and interpolate u' from the values $(u^{(j)})_{j=1}^{n}$ as the value at zero of a polynomial of degree $2t$. This and all following interpolations must be done securely in the sense defined above, i.e. check that all received shares lie on a polynomial of correct degree and abort the protocol if any checks fail. Broadcast your value of u' to all parties.

Round 3: reconstructing u^*. Receive values $u^{'(j)}$ from all other players and interpolate u^* from these values. All parties now hold a common value u^* which is the product of the secrets r^* and $\bar{x}^*$ defined by the shares r and $\bar{x}$ respectively. Compute your public key share $X \leftarrow G^{r/u^*}$ and broadcast this to all parties.

Round 4: public key. Receive shares $X^{(j)}$ from all parties and interpolate X^* from these values. X^* is the public key. We repeat that interpolation must be done securely, i.e. checking all other shares against the subset used for interpolation.

To generate the two public keys for DLIN encryption (X^* and Y^* in the notation of this section) the respective rounds of the two protocols can be combined, giving the same communication cost (number of messages) as for a single public key.

3.5 Efficiency and Security

Efficiency. It is well-known that MPC can in theory be used to compute any functionality yet in practice, the resulting protocols are too slow to be usable, usually due to a massive communication overhead. Our MPC protocol has a communication cost (in number of rounds or messages sent) equivalent to one single MPC multiplication, even for both public keys X and Y since they can be computed in parallel. This is definitely efficient enough to be run in practice: the cost of using MPC for the setup is dwarfed by the cost of malleable proofs so we expect key generation to account for only a small proportion of the running time of the whole protocol. Moreover, the PRSS setup which is the most expensive part of the setup can be run once for a group of parties and the PRSS obtained can then be re-used for many elections, generating new keys using fresh labels

[4] All operations take place in the ring $\mathbb{Z}_q$ so "mod q" is implicit in any operation.

each time. Further, setup is a much less time-critical operation than tallying in a typical deployment of a voting scheme. Therefore, we omit a full analysis of the computational cost of the setup protocol.

Security. Textbook MPC theory says that our MPC protocol is secure against passive adversaries, i.e. who do not send false values during the protocol. However, the simple nature of our protocol together with the security threshold $t < n/3$ yields active security for free. The only operations which parties perform on values that they have received from other, potentially malicious parties are interpolations of polynomials of degree at most $2t$. Therefore, since for each such interpolation there are at least $2t + 1$ correct shares, the malicious parties cannot send incorrect values without causing the protocol to abort. It is important that all interpolations are done securely, i.e. after computing the desired value from any set S of $t+1$ (or $2t+1$) shares it must be verified that all further shares lie on the polynomial defined by the shares in S of the correct degree-bound. We do not care about resilience of the setup protocol against malicious parties causing the protocol to abort: in an election scenario, if a decryptor is caught cheating during key generation then one will probably want to choose a new decryptor and re-run the whole setup.

3.6 Threshold Decryption

To decrypt a ciphertext (A, B, C), each decryptor P_i holding shares $\bar{x}_i$ and $\bar{y}_i$ publishes $D_i = A^{\bar{x}_i} B^{\bar{y}_i}$. This is again a degree-t Shamir-share of $D = A^{1/x} B^{1/y}$. To combine decryption shares and complete a decryption, one interpolates D securely from any $t + 1$ shares $(D_i)_i$ (i.e. checks that all further shares lie on the polynomial defined by the ones used to decrypt). This secure interpolation ensures correctness of the decrypted result if $t < n/3$. On the other side, at the current state, correctness of the election result cannot be ensured against thresholds $t \geq n/3$. As a consequence, zero-knowledge proofs do not provide any benefit at decryption time, which allows us to discard them at decryption time.

If any shares appear incorrect then one can isolate the incorrect shares using Reed-Solomon decoding [21] and still recover the correct decryption as long as $t < n/3$, so up to t malicious decryptors can neither cause a false result to be announced nor prevent the correct result from being computed. This is a significant improvement of the efficiency of the decryption process compared to the original CKLM13 scheme (in which it is proposed using another round of malleable proofs) and could be applied to other voting schemes as well.

4 Computational Analysis of the Proofs in CKLM13

In this section we analyze the computational cost of the malleable proofs underlying the mixnet in the CKLM13 scheme [10]. Since Chase et al. only give an abstract description of their proofs we need to make a reasonable choice of a concrete setting in which to instantiate them. The only known implementation that yields somewhat efficient malleable proofs is that of Groth-Sahai (GS) proofs [9] in a pairing group; this is again an abstract concept for which we need to choose specific groups.

4.1 Choice of Setting

Elliptic curves form the basis of all known implementations of pairing groups which are widely believed to have cryptographic security properties. For such groups, the relevant parameters are q, the logarithm of the group size (roughly: the *bit length* of group elements) and k, the *embedding degree* of the group [22]. As a rule of thumb, the cost of operations in such a group is proportional to q^2 whereas security is proportional to $q \cdot k$; a rough estimate is that for given q, k, the security level is equivalent to a $qk/24$ bit symmetric key. It is clear that choosing k as large as possible results in the greatest efficiency at a desired security level. The parameter k is determined by details of the construction of the underlying elliptic curve; the best known choice is a Barreto-Naehrig (BN) curve [23] which achieves $k = 12$. For this reason, BN curves are the standard choice for implementing pairing-based cryptography nowadays. In this case, to get the equivalent of 128-bit security [24] requires group elements of bit-length $q = 256$ bits.

Choosing BN curves gives an *asymmetric* pairing group, i.e. a triple of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that no efficient homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ are known in either direction[5]. However, the CKLM13 scheme is given in a *symmetric* setting where $\mathbb{G}_1 = \mathbb{G}_2$ so to deploy it on a BN curve requires some modifications that are well established in the literature. Despite the cost of additional equations incurred in the transformation from symmetric to asymmetric settings, the resulting asymmetric protocols usually greatly outperform their symmetric ancestors. We therefore choose to analyse the cost of the CKLM13 malleable proofs in a $q = 256$ bit BN curve with the necessary modifications to the protocol.

For CKLM13, we require two modifications. First, instead of the DLIN assumption (which only applies to a single group), we require what is technically known as SDLIN (symmetric DLIN) [25], the assumption simply states that DLIN holds in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of the setting. This is commonly believed to be the case in groups derived from elliptic curves and the switch from DLIN to SDLIN does not change the protocol. Secondly, since we are using an asymmetric setting, any group element that appears both in groups $\mathbb{G}_1$ and $\mathbb{G}_2$ in the symmetric protocol needs to be replaced by a pair of elements in the asymmetric protocol and "guarded" by an additional equation in the proof. This technique is standard in converting pairing-based schemes from the symmetric to the asymmetric setting.

[5] All three groups are in fact isomorphic but security stems in part from the fact that no efficient way to compute isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ is known.

4.2 Overview of Groth-Sahai Proofs

Groth-Sahai (GS) proofs are based on pairing groups and can be instantiated under several assumptions for several types of equations. We assume an initial set of parameters is given that describe groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of some order p a prime or prime power, with generators (G_1, G_2, G_T) respectively and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. This setting is provided by BN curves; we can abstract away any further details of the curves for the moment. Of interest to us are so-called Pairing Product Equations (PPE) under the SDLIN security assumption. A PPE is an equation with vectors of variables $\underline{\boldsymbol{a}}$ over $\mathbb{G}_1$ and $\underline{\boldsymbol{b}}$ over $\mathbb{G}_2$ of the form

$$\boldsymbol{v} \bullet \underline{\boldsymbol{b}} \;\cdot\; \underline{\boldsymbol{a}} \bullet \boldsymbol{w} \;\cdot\; \underline{\boldsymbol{a}} \bullet \mathit{\Gamma} \bullet \underline{\boldsymbol{b}} = t$$

where $\cdot$ is the group operation in $\mathbb{G}_T$ and $\bullet$ is a scalar product over the pairing, i.e. $\boldsymbol{a} \bullet \boldsymbol{b} := \prod_i e(a_i, b_i)$ and $\boldsymbol{a} \bullet \mathit{\Gamma} \bullet \boldsymbol{b} := \prod_i \prod_j e(a_i, b_j)^{\mathit{\Gamma}_{ij}}$. $\boldsymbol{v}, \boldsymbol{w}$ and t are constants in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

A GS proof proves that the prover knows an assignment of values to a number of variables which satisfies a set of equations. These values are often known as a *witness.* The prover starts by making a *commitment* to each value and then produces a *proof pair*[6] of elements for each equation. The entire proof consists of a commitment for every variable appearing in the equations and a proof pair for each equation. *Verifying* a GS proof involves evaluating a verification equation for each given equation involving the commitments to the variables, the constants in the original equation and the proof pair.

Mathematical overview. In this section we give some of the mathematical ideas necessary to understand how our costing of GS proofs works; the reader can skip the mathematical overview if so inclined without missing the essence of our paper.

The SDLIN GS proofs [25] use modules $B_1 := (\mathbb{G}_1)^3, B_2 := (\mathbb{G}_2)^3, B_T := (\mathbb{G}_T)^9$ that can be seen as groups of vectors and matrices over the original groups and inherit a bilinear pairing $e_B : B_1 \times B_2 \to B_T$. Thus, a basic operation (addition or multiplication) in a module costs 3 respectively 9 operations in the underlying group; the pairing e_B costs 9 e-pairings.

All variables must be committed to; for the vector $\underline{\boldsymbol{a}}$ over $\mathbb{G}_1$ this is done by picking a matrix R_1 of random integers modulo p and computing commitments $\boldsymbol{c} \leftarrow \iota(\underline{\boldsymbol{a}}) + R_1 \cdot U_1$ where ι is an inclusion map from $\mathbb{G}_1$ to B_1 and U_1 is a matrix of constants defined in the setup information. The process for $\underline{\boldsymbol{b}}$ over $\mathbb{G}_2$ is analogous.

A GS proof of a PPE in the DLIN setting is a pair $(\theta, \pi) \in (B_1)^3 \times (B_2)^3$ computed according to the following equations.

$$\pi \;\leftarrow\; R_1^\top \iota_2(\boldsymbol{w}) + R_1^\top \mathit{\Gamma} \iota_2(\underline{\boldsymbol{b}}) + R_1^\top \mathit{\Gamma} R_2 U_2 - T^\top U_2 \qquad (\mathit{\Pi})$$

$$\theta \;\leftarrow\; R_2^\top \iota_1(\boldsymbol{v}) + R_2^\top \mathit{\Gamma}^\top \iota_1(\underline{\boldsymbol{a}}) + T U_1 \qquad (\mathit{\Theta})$$

[6] This is our terminology. Such a pair is commonly just called a “proof” but we wish to distinguish between the elements associated with a particular equation and the proof as a whole.

Here R_1, R_2 are the random elements used to commit to elements in $\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}}$ respectively, T is a matrix of random integers modulo p chosen to randomise the proof of this PPE and U_1, U_2 are matrices of constants defined in the setup information.

Verification of such a proof involves checking the following equation. Here $\boldsymbol{c}, \boldsymbol{d}$ are the commitments to $\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}}$ respectively and $\otimes$ is the scalar product over the pairing e_B in the B-modules.

$$\iota_1(\boldsymbol{v}) \otimes \boldsymbol{d} \ \cdot \ \boldsymbol{c} \otimes \iota_2(\boldsymbol{w}) \ \cdot \ \boldsymbol{c} \otimes \Gamma \boldsymbol{d} \overset{?}{=} \iota_T(t) \ \cdot \ U_1 \otimes \pi \ \cdot \ \theta \otimes U_2 \qquad \text{(V)}$$

To count the number of operations in the CKLM13 scheme, we must deal with several small issues. First, the original paper describes the protocol in terms of a set of equations that are "almost" PPEs — almost, because they use abbreviations in their notation and our first step is to expand these into actual PPE that can be processed by the GS proof system. Secondly, we make the necessary changes to deploy the protocol in an asymmetric setting. Thirdly, starting with the equations to create and verify proofs of PPE we optimise them for the specific equations in CKLM13, i.e. we remove terms that cancel out or have all-zero coefficients.

4.3 Results

We let L be the number of votes shuffled in a run of the mixnet. Of the $4L$ variables and 11 equations given in CKLM13, equations 1–4 are simple PPE, 5 and 6 together require L supporting variables and equations to expand into a full PPE, 7 and 8 are linear PPE, 9–11 are quantified $(\forall i : 1 \leq i \leq L)$ so are in fact L PPE each. To map these into an asymmetric setting requires another $2L$ supporting variables and $4L$ supporting equations. All together we end up with $8 + 8L$ equations of which the first 8 have L-fold products each and the remaining $8L$ have only constant-size products; in total we have $4L$ variables in $\mathbb{G}_1$ and $7L$ in $\mathbb{G}_2$.

Analysing the equations and taking into account components that have all-zero coefficients (which therefore contribute nothing to the cost of computation), we find an upper bound on the computation cost as presented in Table 1. For our detailed calculations we refer to Appendix C.

Table 1. Number of elementary operations in proof creation and verification in CKLM13.

Task	$\mathbb{G}_1$ mult.	$\mathbb{G}_2$ mult.	$\mathbb{G}_1$ op.	$\mathbb{G}_2$ op.	$\mathbb{G}_T$ op.	Pairing
Create proof	$163L + 72$	$183L + 72$	$134L + 48$	$167L + 48$		
Verify proof					$657L + 252$	$657L + 324$

For the remainder of this work, we refrain from considering group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, because these are about the factor *bit-length of group elements* faster than multiplications [26] and consequently do not influence the

Table 2. Operation timings for Barreto-Naehrig curve over 256-bit prime fields with embedding degree 12 with the MIRACL library and the Beuchat et al. implementation.

Elementary Operation	MIRACL	Beuchat et al.
$\mathbb{G}_1$ Multiplication	0.22 ms	n.a.
$\mathbb{G}_2$ Multiplication	0.44 ms	n.a.
Pairing	2.32 ms	0.39 ms

feasibility analysis significantly. The following formula allows us to estimate the running time of an individual mix node and the voter's verification:

$$\begin{aligned} s(L) = {} & (163 \times L + 72) \times \mathbb{G}_1 \text{ Multiplication Time } + \\ & (183 \times L + 72) \times \mathbb{G}_2 \text{ Multiplication Time } + \\ & (657 \times L + 324) \times \text{ Pairing Time} \end{aligned}$$

Optimisations. We stress that our results are only an upper bound on the cost of computing a CKLM13 proof as there are several feasible optimisations that we have not yet considered. More details on possible optimisations are included in our detailed analysis in Appendix C.

Timings. To the best of our knowledge, the MIRACL cryptography C library [12] is the most established open-source library to support BN curves. Recent timings taken on a 2.4 GHz Intel i5 520M processor [27] lead to the results provided in Table 2 (second column). The fastest claimed results for pairings on 256-bit BN curves which we are aware of are from Beuchat et al. [28, 29] who compute a pairing in 0.39 ms, compared to 2.32 ms currently achievable with MIRACL. However, times for multiplications using their implementation are not available (ref. to Table 2 third column). As pairings are the most expensive operation in the CKLM13 scheme, in the following, we hypothetically assume Beuchat et al.'s pairing time of 0.39 ms, while all other costs are assumed to be equal to MIRACL. The hypothetical timings for Beuchat et al.' implementation are considered in parallel to the MIRACL timings.

To bring these numbers into relation to real-world elections, we consider cryptographic voting as substitution for postal voting in the German case. In the German federal election 2009, 62.2 millions citizens were eligible to vote [30]. On average, electoral districts in Darmstadt, Germany have a size of 1100 voters, while for postal votes on average 3.5 electoral districts are aggregated. In 2009, 21.4% of the eligible voters cast their vote via postal voting. Considering all eligible voters, this results in 13.3 millions postal votes, while for each postal voting district in Darmstadt, Germany, this results in 824 postal votes. Table 3 summarizes the expected running times for one mix node both for the MIRACL library and the hypothetical Beuchat et al. implementation. It should be noted that the running time for the voter's verification is close to the running time of an individual mix node. These timings show that malleable proof based cryptographic voting schemes are feasible for a moderate number of voters, however their efficiency does not compare with Wikström's work or Bayer and Groth's work that achieve mix proofs and their verification for 100.000 ElGamal ciphertexts in around 2 minutes [3, 6].

Table 3. Expected running times for individual mix nodes in the CKLM13 scheme with different numbers of voters.

Number of voters	MIRACL	Beuchat et al.
10	17.2 s	3.9 s
824 (Electoral District)	$\approx 22,5$ min	≈ 5.1 min
1.000	$\approx 27,3$ min	≈ 6.2 min
100.000	≈ 45 h	≈ 10 h
10.000.000	≈ 190 d	≈ 43 d
13.300.000 (German Federal Election)	≈ 252 d	≈ 57 d

5 Conclusion

In this work, we build upon the CKLM13 cryptographic voting scheme [10], which is based on the concept of malleable proofs invented in [7]. As opposed to existing mix-based approaches, CKLM13 allows to generate verification data which is independent of the number of mix nodes and the number of decryptors involved in the tallying process. However, so far the theoretical innovations are far from practical use. To bridge the gap between innovation and practice, in this paper, we have addressed two crucial questions which remain open in CKLM13. First, we propose a distributed key generation protocol for the DLIN encryption scheme based upon a multi-party computation protocol due to Smart and Geisler [11] and therefore succeed in ensuring security against up to $n/3$ misbehaving participants. By construction of the protocol, we do not achieve security against $n/3$ or more dishonest decryptors whatever happens in the decryption phase; for fewer than $n/3$ dishonest decryptors however the correctness of the election result is verifiable even without any proofs of correct decryption, allowing us to omit them for the time being. Secondly, we investigate CKLM13 in detail and identify elementary operations underlying their constructions. We count the number of such operations used for a single mix node. Based on timings from the MIRACL library, we calculate the running time for single mix nodes, which is almost the same as the running time of a voter verifying the election result.

We base our conclusion upon data obtained from the German Federal election in 2009. It turns out that the replacement of postal voting by cryptographic voting based on malleable proofs would be feasible on an electoral district level. Assuming that three mix nodes are in place with an average number of 824 absentee voters, tallying the election and the voter's verification of the result can be finalized in 90 minutes. This corresponds to the time needed to tally the postal votes in Darmstadt, Germany [31]. However, the results obtained in this work also show that the application of large-scale malleable proof based cryptographic voting is not feasible today. The tallying process on a city level (100.000 eligible voters) would require more mix nodes to be involved. Considering malleable proof based cryptographic voting on city level with five mix nodes and the voter's verification would result in a running time close to two weeks.

We guide future research in several directions: The constructed distributed key generation protocol for the DLIN encryption scheme is based on the assumption that $t < n/3$ participants are actively cheating (for both privacy and verifiability). Privacy against up to $t < n/2$ cheating administrators should be possible with standard MPC techniques. According to Smart [32], implementing our key generation protocol on top of SPDZ [33], for which a practical implementation exists, should even give privacy and verifiability against up to $t = n - 1$ cheaters [32]. For verifiability against even n out of n cheating decryptors, we believe that this is achievable more cheaply than by using another round of malleable proofs by exploiting the pairing operation directly, but leave this idea for future work. Either way, the malleable proofs in the mixnet constitute the dominating cost of the CKLM13 protocol (should one wish to use malleable proofs of correct decryption, these can be based on a much simpler set of GS equations). This justifies our choice to restrict our formal analysis of computational costs in the CKLM13 protocol to the mixing phase. Even though carefully designed, we leave the analysis and the correctness proof of the constructed protocol as a task for future work.

The feasibility estimations of this work are an upper bound on the real cost in a full implemented version of the modified CKLM13 scheme. For instance, expressions of the form $\sum_{i=1}^{l} v_i X_i$ in a group $\mathbb{G}_j, j \in \{1, 2\}$ we counted as l products and $l - 1$ sums, yet algorithms exist [22] to perform such operations more efficiently. One might also consider applying batch techniques [34] because proofs have large numbers of equations of very similar form. Thereby, the number of pairings required to verify a proof might be significantly reduced. Finally, there exist cryptographic libraries providing better performance than MIRACL. The works of Beuchat et al. [28, 29] show that pairing times can be reduced to 1/5 of the MIRACL timings, which would speed up the mixing and verification process by a factor 5. For the future, Beuchat et al.'s implementation should be extended towards a full cryptographic library such that ultimately an cryptographic voting scheme based on malleable proofs can be deployed.

Acknowledgements. The authors would like to thank Dr. Essam Ghadafi for comments on Groth-Sahai proofs and suggesting some optimisations, Prof. Nigel P. Smart and Dr. Ashish Choudhary for comments on multi-party computation, Markulf Kohlweiss for comments on the design of the CKLM12/13 protocols and Dr. Hugo Jonker for organising a PhD workshop on voting in 2012 at which the authors of this paper met and came up with the ideas for this paper. Finally, the authors would like to thank the anonymous reviewers for their helpful comments.

This work has been supported in part by EPSRC via grant EP/H043454/1, in part by the German Science foundation (DFG) via the project "ModIWa2 - Juristisch-informatische Modellierung von Internetwahlen", and in part by the

German Federal Ministry of Education and Research (BMBF) via the project "BoRoVo - BoardRoomVoting".

References

1. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2) (1981) 84–90
2. Wikström, D.: A commitment-consistent proof of a shuffle. In: ACISP. Volume LNCS 5594., Springer (2009) 407–421
3. Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: AFRICACRYPT. (2010) 100–113
4. Wikström, D.: A commitment-consistent proof of a shuffle. Cryptology ePrint Archive, Report 2011/168 (2011) `http://eprint.iacr.org/`.
5. Lipmaa, H., Zhang, B.: A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. IACR Cryptology ePrint Archive (2011) 394
6. Bayer, S., Groth, J.: Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In: Eurocrypt 2012. (2012) 263–280
7. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable Proof Systems and Its Applications. In: Eurocrypt 2012. Volume LNCS 7237., Springer (2012) 281–300
8. Boneh, D., Boyen, X., Schacham, H.: Short Group Signatures. In: Advances in Cryptology (Crypto 2004). Volume LNCS 3152., Springer (2004) 41–55
9. Groth, J., Sahai, A.: Efficient Non-interactive Proof Systems for Bilinear Groups. In: Eurocrypt 2008. Volume LNCS 4965., Springer (2008) 415–432
10. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Verifiable Elections That Scale for Free. In: Proceedings of Practice and Theory in Public Key Cryptography - PKC 2013. Volume LNCS 7778., Springer (2013) 479–496
11. Geisler, M., Smart, N.P.: Distributing the Key Distribution Centre in Sakai-Kasahara Based Systems. In: Cryptography and Coding 2009. Volume LNCS 5921., Springer (2009) 252–262
12. CertiVox: MIRACL Crypto SDK. `https://certivox.com/solutions/miracl-crypto-sdk/` Online; accessed 22 March, 2013.
13. Fouque, P.A., Poupard, G., Stern, J.: Sharing Decryption in the Context of Voting or Lotteries. In: Proceedings of the 4th International Conference on Financial Cryptography (FC 2000). Volume LNCS 1962., Springer (2001) 90–104
14. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979) 612–613
15. Cramer, R., Damgård, I., Ishai, Y.: Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In: Proceedings of TCC 2005. Volume LNCS 3378., Springer (2005) 342–362
16. Pedersen, T.: A Threshold Cryptosystem without a Trusted Party. In: Eurocrypt 1991. Volume LNCS 547., Springer (1991) 522–526
17. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Eurocrypt 1999. Volume LNCS 1592., Springer (1999) 295–310
18. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty Computation, an Introduction. `http://www.cs.au.dk/~jbn/smc.pdf` Online; accessed 22 March, 2013.
19. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989, ACM (1989) 201–209

20. Kate, A., Goldberg, I.: Asynchronous distributed private-key generators for identity-based cryptography. IACR Cryptology ePrint Archive (2009) 355
21. McEliece, R.J., Sarwate, D.V.: On sharing secrets and Reed-Solomon codes. Communications of the ACM **24** (1981) 583–584
22. Cohen, H., Frey, G., eds.: Handbook of elliptic and hyperelliptic curve cryptography. CRC Press (2005)
23. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: SAC 2005. Volume LNCS 3897., Springer (2006) 319–331
24. European Network of Excellence in Cryptology II: Ecrypt II Yearly Report on Algorithms and Key Sizes. `http://www.keylength.com/en/3` Online; accessed 22 March, 2013.
25. Ghadafi, E., Smart, N.P., Warinschi, B.: Groth-Sahai Proofs Revisited. In: Proceedings of Practice and Theory in Public Key Cryptography - PKC 2010. Volume LNCS 6056., Springer (2010) 177–192
26. Smart, N.P.: Personal communication
27. CertiVox: CertiVox Wiki, Benchmarks and Subs. `https://wiki.certivox.com/display/EXT/Benchmarks+and+Subs` Online; accessed 22 March, 2013.
28. Beuchat, J.L., Diaz, J.E.G., Mitsunari, S., Okamoto, E., Rodriguez-Henriquez, F., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. In: Cryptology ePrint Archive, Report 2010/354. (2010)
29. Mitsunari, S.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. `http://homepage1.nifty.com/herumi/crypt/ate-pairing.html` Online; accessed 22 March, 2013.
30. Der Bundeswahlleiter: Pressemitteilung 16. Februar 2009
31. Citizens Registration Office, D.E.M.N.: Personal communication
32. Smart, N.P.: Personal communication
33. Damgard, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority — or: Breaking the SPDZ Limits. In: Cryptology ePrint Archive, Report 2012/642. (2012)
34. Blazy, O., Fuchsbauer, G., Izabachène, M., Jambert, A., Sibert, H., Vergnaud, D.: Batch Groth-Sahai. In: ACNS 2010. Volume LNCS 6123., Springer (2010) 218–235
35. Knuth, D.: The Art of Computer Programming. Volume Volume 4A. Addison-Wesley Professional (2011)

A Mathematical Background

A.1 Generating Subsets

In our MPC protocol, to set up the PRSS participants need to iterate over all subsets $S \subseteq \{1, \ldots, n\}$ of size $|S| = n - t$ that contain a given index i.

The following algorithm from Knuth [35, 7.2.1.3 Alg. T] enumerates all subsets of size t of $\{1, \ldots, n\}$. The command **yield** outputs the next item but does not return from the function.

```
procedure combinations($n, t$)
100  $c \leftarrow []$
101  for $k \leftarrow 1, \ldots, t$ do
102      $c[k] \leftarrow k - 1$
103  $c[t+1] \leftarrow n; c[t+1] \leftarrow 0$
```

```
$x \leftarrow 0; j \leftarrow t$
**loop**
    $item \leftarrow []$
    for $k \leftarrow 1, \dots, t$ do
        $item[k] \leftarrow c[t-k+1]+1$
    if $j > 0$ then
        $x \leftarrow j$
    else
        if $c[1]+1 < c[2]$ then
            $c[1]++$
            **yield** $item$
        else
            $j \leftarrow 2$
        $c[j-1] \leftarrow j-2$
        $x \leftarrow c[j]+1$
        **while** $x == c[++j]$ do
            $c[j-1] = j-2$
            $x \leftarrow c[j]+1$
        if $j > t$ then
            **yield** $item$
            **halt**
        $c[j] \leftarrow x$
        $j--$
```

A.2 Polynomial Interpolation

Let $\mathbb{F}$ be a field. A degree-t-bounded polynomial p over $\mathbb{F}$ is defined uniquely by $t+1$ "points" that can be

- Points (x, y) s.t. $p(x) = y$.
- Coefficients c_i in the representation $p(x) = \sum_{i=0}^{t} c_i x^i$.

Converting from coefficients to points is called *evaluating* the polynomial, converting from points to coefficients or finding the value at a new point given a set of existing ones is called *interpolating* the polynomial.

Given $t+1$ points $(x_i, y_i)_{i=0}^{t}$ with $x_i \neq x_j$ for $i \neq j$, the unique polynomial p of degree at most t through these points can be epxressed as $p(x) = \sum_{i=0}^{t} L_i(x) \cdot y_i$ where the l_i are the *Lagrange coefficients*

$$l_i(x_i) = \prod_{j=0, j\neq i}^{t} \frac{x - x_j}{x_i - x_j}$$

A common case in secret sharing schemes is interpolating the secret as a value at 0 given shares s_i representing values at given points i in a set S of the right size. In this case the interpolation formula reduces to

$$p(0) = \sum_{i \in S} s_i \cdot \prod_{j \in S, j \neq i} \frac{(-j)}{i - j}$$

We can see that this is a *linear* function in the inputs s_i. This allows us to lift polynomial interpolation and secret sharing from $\mathbb{F}_q$ into cyclic groups of order q (actually, into any vector spaces over $\mathbb{F}_q$): If $\phi(s_1, \ldots, s_{t+1})$ is the interpolation function over $\mathbb{F}_q$ then for the corresponding function Φ on a group G of order q (replacing $+$ by the group operation and $\cdot$ by group exponentiation),

$$\Phi(G^{a_1}, \ldots, G^{s_{t+1}}) = G^{\phi(s_1, \ldots, s_{t+1})}$$

A.3 Polynomials with a Zero-Set

For a set $S \subseteq \{1, \ldots, n\}$ of size t, let $p_{0:S}$ be the unique polynomial of degree at most t such that $p_{0:S}(s) = 0$ on all $s \in S$ and $p_{0:S}(0) = 1$. To evaluate this polynomial at a point $x \notin S$, one computes

$$p_{0:S}(x) = \prod_{s \in S} (x - s) \Big/ \prod_{s \in S} (-s)$$

Since our application requires many evaluations of such polynomials at different points, one can precompute the denominator.

B Shamir's Secret Sharing and Applications

B.1 Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme [14] allows a person to share a secret x among n people such that any $t+1$ of them can reconstruct the secret but any coalition of t or fewer gain no information on the secret. This is done in a finite field of size $q > n$ by assigning each shareholder an index i (indices are public); the secret-holder computes a random degree-bound t polynomial p such that $p(0) = x$ i.e. lets $c_0 = x$, picks $c_1, \ldots, c_t \twoheadleftarrow \mathbb{F}_q$ and $p(x) := \sum_{j=0}^{t} c_j x^j$. Each shareholder then receives $p(i)$ as their share where i is their index. To reconstruct the secret, any $t+1$ shareholders can interpolate $p(0)$, for example using Lagrange coefficients: for a set S of $t+1$ indices,

$$p(0) = \sum_{s \in S} p(s) \cdot \prod_{j \in S, j \neq s} \frac{(-j)}{s - j}$$

B.2 PRSS

Pseudorandom secret sharing (PRSS) [15] is an extension of Shamir's scheme that allows a group of parties to jointly generate shares of a secret without any one party (or indeed a subset of less than t parties) needing to know the secret. PRSS operates in two phases and uses a pseudorandom function. In the first phase, every subset S of $n-t$ parties communicates to jointly generate a shared secret k_S. In the second phase, all parties can generate shares of many fresh secrets locally, i.e. without any further communication, by computing a function of a public label and their keys k_S for all the sets S that they are part of. The result is that for each label l, all parties obtain a degree-t share of a secret x^*_l. This step is essentially a polynomial interpolation using a pseudorandom function on the secret keys and the public label. For the ElGamal encryption scheme, one can turn the key generation algorithm into a protocol using a PRSS directly.

Preliminaries. We assume that $\mathbb{G}$ is a group of order a prime power q with generator G and that $F : \mathbb{Z}_q \times L \to \mathbb{Z}_q$ is a pseudorandom function, where L is some set of labels. We consider a set of n parties with indices $i \in \{1, \ldots, n\}$ that wish to generate degree-t-bounded Shamir shares and describe the protocol for party i.

Setup phase. For each set [7] $S \subseteq \{1, \ldots, n\}$ of size $|S| = n - t$ containing your own index i, generate a random value $r_S \twoheadleftarrow \mathbb{Z}_q$ and distribute it securely to the other members of S. For each such set S, obtain the random values $r_S^{(j)}$ from all other members of S and add them to create a key $k_S \leftarrow \sum_{j \in S} r_S^{(j)}$. The result of this step is that each set S of $n - t$ parties now share a secret key k_S; the communication and computation cost is proportional to $\binom{n}{t}$.

Drawing values. All players can now draw values from the PRSS given a label $l^* \in L$ as follows: for each set S of size $n - t$ containing your index, compute the (unique) degree-t polynomial p_S such that $p_S(0) = 1$ and $p_S(j) = 0$ for all $j \in \{1, \ldots, n\} \setminus S$. Your share x for label l^* is

$$x(l^*) = \sum_{\substack{S \subseteq \{1,\ldots,n\} \\ |S|=n,\ i \in S}} p_S(i) \cdot F(k_S, l^*)$$

If all players draw a value from the PRSS for a fresh common label l^*, the result is that each P_i holds a degree-t share $x(l^*)$ of a fresh secret $x^*(l^*)$.

For the special case $t = n - 1$ there is a shortcut. Each P_i can simply pick a random value x_i as her share, these n values trivially define a polynomial of degree-bound $n - 1$.

C Efficiency Analysis of Groth-Sahai Proofs in CKLM13

The malleable proofs for the mixnet in CKLM13 are based on the DLIN assumption in a *symmetric* pairing setting and use the following set of equations [10, Page 11], where we have underlined the witnesses:

[7] How to enumerate these sets efficiently is described by Knuth [35] and given in Appendix A.1.

$$\prod_{i=1}^{L} e\left(\underline{a_i}, u'_i\right) = \prod_{i=1}^{L} e\left(\underline{a_i}, \underline{f^{r'_i}}\right) e\left(g_i, u_i\right) \tag{1}$$

$$\prod_{i=1}^{L} e\left(\underline{b_i}, u'_i\right) = \prod_{i=1}^{L} e\left(\underline{b_i}, \underline{f^{r'_i}}\right) e\left(\gamma_i, u_i\right) \tag{2}$$

$$\prod_{i=1}^{L} e\left(\underline{a_i}, v'_i\right) = \prod_{i=1}^{L} e\left(\underline{a_i}, \underline{h^{s'_i}}\right) e\left(g_i, v_i\right) \tag{3}$$

$$\prod_{i=1}^{L} e\left(\underline{b_i}, v'_i\right) = \prod_{i=1}^{L} e\left(\underline{b_i}, \underline{h^{s'_i}}\right) e\left(\gamma_i, v_i\right) \tag{4}$$

$$\prod_{i=1}^{L} e\left(\underline{a_i}, w'_i\right) = \prod_{i=1}^{L} e\left(\underline{a_i}, \underline{g^{r'_i} g^{s'_i}}\right) e\left(g_i, w_i\right) \tag{5}$$

$$\prod_{i=1}^{L} e\left(\underline{b_i}, w'_i\right) = \prod_{i=1}^{L} e\left(\underline{b_i}, \underline{g^{r'_i} g^{s'_i}}\right) e\left(\gamma_i, w_i\right) \tag{6}$$

$$\prod_{i=1}^{L} \underline{a_i} g_i^{-1} = 1 \tag{7}$$

$$\prod_{i=1}^{L} \underline{b_i} \gamma_i^{-1} = 1 \tag{8}$$

$$\forall i \leq L \quad e\left(\underline{a_i}, \underline{a_i}\right) = e\left(g, \underline{b_i}\right) \tag{9}$$

$$\forall i \leq L \quad e\left(\underline{f^{r'_i}}, g\right) = e\left(f, \underline{g^{r'_i}}\right) \tag{10}$$

$$\forall i \leq L \quad e\left(\underline{h^{s'_i}}, g\right) = e\left(h, \underline{g^{s'_i}}\right) \tag{11}$$

C.1 Additive versus Multiplicative Notation.

A group $\mathbb{G}$ comes with a group operation $\mathbb{G} \times \mathbb{G} \to \mathbb{G}$ and the universal operation of group exponentiation $\mathbb{Z} \times \mathbb{G} \to \mathbb{G}$. Groups can be written either additively, the group operation being denoted $G+H$ and exponentiation $n \cdot G$, nG or sometimes [25] $[n]G$, or multiplicatively in which case the operation is written $G \cdot H$ or simply GH and exponentiation G^n. The notation varies between papers: Chase et al. [10] write all groups multiplicatively whereas we stick with the notation proposed by Groth and Sahai [9] in which $\mathbb{G}_1$ and $\mathbb{G}_2$ are written additively and $\mathbb{G}_T$ multiplicatively[8].

[8] The reason for this convention is that $\mathbb{G}_1$ and $\mathbb{G}_2$ are usually implemented as subgroups of the group of points on an elliptic curve where the operation is "point addition"; $\mathbb{G}_T$ by contrast typically is a subgroup of $(\mathbb{Z}_q^*, \times)$. The astute reader may have already noticed our use of this convention in the formula $e(aG_1, bG_2) = e(G_1, G_2)^{ab}$ above.

We stress that all these conventions are simply a matter of notation and do not change the nature of the actual operations in the groups themselves.

C.2 Expanding the Equations

The above notation contains several "abbreviations". Groth and Sahai [9] give us a template for proofs of pairing-product equations of the following form[9], where variables are underlined.

$$\prod_{i=1}^{n} e\left(v_i, \underline{b_i}\right) \cdot \prod_{j=1}^{m} e\left(\underline{a_i}, w_i\right) \cdot \prod_{i=1}^{n}\prod_{j=1}^{m} e\left(\underline{a_i}, \underline{b_i}\right)^{\gamma_{ij}} = t \tag{PPE}$$

Equations 1–4 can be cast into the PPE schema by inverting the u_i' and u_i, swapping sides, letting the constants $v_i = 0 (\forall i)$ and setting $\gamma_{ij} = \delta_{ij} := (1$ if $i = j$ else $0)$, i.e. Γ expressed as a matrix is the identity matrix. For example, equation 1 becomes (switching to additive notation in $\mathbb{G}_1, \mathbb{G}_2$ too)

$$\prod_{i=1}^{L} e\left(0, \underline{r_i' f}\right) \prod_{i=1}^{L} e\left(\underline{a_i}, -u_i'\right) \prod_{i=1}^{L}\prod_{j=1}^{L} e\left(\underline{a_i}, \underline{r_j' f}\right)^{\delta_{ij}} = \prod_{i=1}^{L} e\left(g_i, -u_i\right) \tag{1'}$$

where the right-hand side is a constant in $\mathbb{G}_T$.

For equations 5 and 6 we have to expand the products[10] of witnesses. For example, to use the expression $\underline{a}+\underline{b}$ in a PPE we introduce a new witness $\underline{c} = \underline{a}+\underline{b}$ that we use in the actual PPE and a new equation (assuming the witnesses are in $\mathbb{G}_2$)

$$e\left(G_1, \underline{a}\right) e\left(G_1, \underline{b}\right) e\left(-G_1, \underline{c}\right) = 1$$

With these new equations, old equations 5 and 6 can be cast as PPE like equations 1–4. For equations 7 and 8, we rewrite the equations as PPE in the asymmetric setting as

$$\prod_{i=1}^{L} e\left(\underline{a_{1,i}}, -G_2\right) = 1 \qquad \prod_{i=1}^{L} e\left(\underline{b_{1,i}}, -G_2\right) = 1 \tag{7', 8'}$$

Equations 9–11 are simple enough to turn into PPE, but note two complications. First, each of these equations is in fact L PPE since there is a quantification over i. Secondly, in an asymmetric setting a witness that is used in both groups needs to be committed to twice, once in each group, and an extra equation is required to ensure that this commitment was done correctly.

For example, to use $e\left(\underline{a_i}, \underline{a_i}\right)$ in an equation requires a commitment to a_i in both groups, two separate variables $a_{1,i}$ and $a_{2,i}$ and the extra equation

$$e\left(G_1, \underline{a_{2,i}}\right) \cdot e\left(\underline{a_{1,i}}, -G_2\right) = 1$$

[9] In the Groth-Sahai paper, the variables were named x and y and the constants a and b which conflicts with the notation of CKLM13. We keep the naming of the variables from CKLM13.

[10] In our notation, sums.

All together we have the following variables.

Variables	Group
$a_{1,i},\, a_{2,i}$	$\mathbb{G}_1,\, \mathbb{G}_2$
$b_{1,i},\, b_{2,i}$	$\mathbb{G}_1,\, \mathbb{G}_2$
$f_1^{r'_i},\, f_2^{r'_i}$	$\mathbb{G}_1,\, \mathbb{G}_2$
$h_1^{s'_i},\, h_2^{s'_i}$	$\mathbb{G}_1,\, \mathbb{G}_2$
$g^{r'_i}$	$\mathbb{G}_2$
$g^{s'_i}$	$\mathbb{G}_2$
$c_i = g^{r'_i} \cdot g^{s'_i}$	$\mathbb{G}_2$

This is a total of $11L$ variables of which $4L$ lie in $\mathbb{G}_1$ and $7L$ in $\mathbb{G}_2$.

Equations 1–6 give us 6 PPE with L-fold products each, equations 7 and 8 are one PPE with an L-fold product each (but a simpler form) and equations 9–11 together with our supporting equations for the variable pairs and to expand products give us $8L$ PPE with 2 or 3-fold products. Our supporting equations, together with new equations 9–11, become (where each equation stands for a sequence of L equations indexed by i):

$$e\left(\underline{a_{i,1}}, \underline{a_{i,2}}\right) e\left(-G_1, \underline{b_{2,i}}\right) = 1 \tag{9'}$$

$$e\left(\underline{r'_{i,1} f}, G_2\right) \cdot e\left(-f, \underline{r'_{i,1} G_2}\right) = 1 \tag{10'}$$

$$e\left(\underline{s'_{i,1} h}, G_2\right) \cdot e\left(-h, \underline{s'_i G_2}\right) = 1 \tag{11'}$$

$$e\left(\underline{a_{1,i}}, G_2\right) \cdot e\left(-G_1, \underline{a_{2,i}}\right) = 1 \tag{12'}$$

$$e\left(\underline{b_{1,i}}, G_2\right) \cdot e\left(-G_1, \underline{b_{2,i}}\right) = 1 \tag{13'}$$

$$e\left(\underline{r'_{i,1} f}, G_2\right) \cdot e\left(-G_1, \underline{r'_{i,2} f}\right) = 1 \tag{14'}$$

$$e\left(\underline{s'_{i,1} h}, G_2\right) \cdot e\left(-G_1, \underline{s'_{i,2} h}\right) = 1 \tag{15'}$$

$$e\left(G_1, \underline{r'_{i,2} G_2}\right) \cdot e\left(G_1, \underline{s'_{i,2} G_2}\right) \cdot e\left(-G_1, \underline{c_i}\right) = 1 \tag{16'}$$

C.3 Counting the Cost

We follow the recipe of Groth and Sahai for creating proofs of PPE and count the cost of operations necessary to produce and verify these proofs.

Preliminaries. In the DLIN setting for Groth-Sahai proofs, the relevant spaces involved are

$$\begin{tikzcd}
A_1 = \mathbb{G}_1 \arrow[d, "\iota_1"'] & \times & A_2 = \mathbb{G}_2 \arrow[d, "\iota_2"'] \arrow[r, "e"] & A_T = \mathbb{G}_T \arrow[d, "\iota_T"'] \\
B_1 = (\mathbb{G}_1)^3 & \times & B_2 = (\mathbb{G}_2)^3 \arrow[r, "e_B"] & B_T = (\mathbb{G}_T)^{3\times 3}
\end{tikzcd}$$

The basic operations we count are additions, multiplications and pairings in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. We assume integer operations are cheap by comparison and disregard them. The inclusion maps are defined as

$$\iota_j : \mathbb{G}_j \to B_j,\ x \mapsto (0, 0, x) \qquad (j \in \{1, 2\})$$

which involve no additions or multiplications so we disregard their cost. We only use ι_T on constants so we can precompute it and ignore its cost too.

In the formulae that follow, we denote a group operation (written additively) in $\mathbb{G}_j$ where $j \in \{1, 2, T\}$ by S_j, a multiplication $\mathbb{Z}_q \times \mathbb{G}_j \to \mathbb{G}_j$ as E_j and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as P.

The corresponding costs in the B-modules are tripled for $j \in \{1, 2\}$ and increased ninefold for $i = T$. A pairing in the B-modules costs $9P$.

Setup. We ignore the cost of the Groth-Sahai setup as it is preformed once before the poll and mixnet in the CKLM13 protocol are run and is therefore less time-critical; the constants included in the setup information are nonetheless important for our purposes.

For the DLIN setting, Groth and Sahai pick constants $U_{j,1}, U_{j,2}, U_{j,3}$ each in B_j for both $j \in \{1, 2\}$ to allow commitment to elements in the respective A_j. (In the original paper, the B_2 elements are called u and the B_1 ones v.) The original description of the DLIN setting is based on symmetric groups where further matrices H_η are required to re-randomise the proofs; since we assume an asymmetric group these are unnecessary.

Commitments. For each variable a in $\mathbb{G}_j$ for $j \in \{1, 2\}$ Groth and Sahai commit to this variable as

$$c = \iota_j(a) \underbrace{+}_{S_{B_j}=3S_j} \underbrace{\sum_{k=1}^{3} r_{j,k} U_{j,k}}_{3(2S_j+3E_j)} \qquad \text{cost: } 9S_j + 9E_j$$

where $(r_{j,k})_k$ is a vector of three randomly chosen integers for this variable and the $U_{j,k}$ are the constants produced during the set-up of the Groth-Sahai scheme.

Proofs. A Groth-Sahai proof of a PPE in the DLIN setting is a list of commitments to the variables and a pair $(\pi, \theta) \in {B_2}^3 \times {B_1}^3$. For each PPE with l-fold products, one draws a matrix $T \twoheadleftarrow \mathbb{Z}_q^{3\times 3}$ of random integers and, together with the vectors of variables $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively and the matrices R_1, R_2 of the random values used to commit to them, the vectors of constants $\boldsymbol{a}, \boldsymbol{b}$ and the matrix Γ, one computes

$$\pi \quad \leftarrow \quad R_1^\top \iota_2(\boldsymbol{w}) + R_1^\top \Gamma \iota_2(\underline{\boldsymbol{b}}) + R_1^\top \Gamma R_2 U_2 - T^\top U_2 \qquad (\varPi)$$

$$\theta \quad \leftarrow \quad R_2^\top \iota_1(\boldsymbol{v}) + R_2^\top \Gamma^\top \iota_1(\underline{\boldsymbol{a}}) + T U_1 \qquad (\varTheta)$$

In equations 1–6, Γ will be the identity matrix so we can simplify these equations before we count operations. Further the constants v_i are 0 for all i so we can omit them too. The lengths of all vectors of variables and constants in these equations are L.

$$\pi \quad \leftarrow \quad R_1^\top \underset{3(LE_2+(L-1)S_2)}{\cdot} \iota_2 \left(\boldsymbol{w} \underset{LS_2}{+} \boldsymbol{b} \right) \underset{3S_2}{+} \left(R_1^\top R_2 - T^\top \right) \underset{3(3E_2+2S_2)}{\cdot} U_2 \tag{$\mathit{\Pi}_1$}$$

$$\theta \quad \leftarrow \quad R_2^\top \underset{3(LE_1+(L-1)S_1)}{\cdot} \iota_1 \left(\boldsymbol{a} \right) \underset{3S_1}{+} T \underset{3(3E_1+2S_1)}{\cdot} U_1 \tag{$\mathit{\Theta}_1$}$$

The total cost of such an equation is therefore $(3L + 9)E_1 + (3L + 9)E_2 + (3L + 6)S_1 + (4L + 6)S_2$.

Equations 7' and 8' are L-fold linear PPE, i.e. $\mathit{\Gamma}$ is all zeroes and even $\boldsymbol{v} = \boldsymbol{0}$; we can simplify the proof pair creation to the following.

$$\pi \quad \leftarrow \quad R_1^\top \underset{3(LE_2+(L-1)S_2)}{\cdot} \iota_2(\boldsymbol{w}) \underset{3S_2}{-} T^\top \underset{3(3E_2+2S_2)}{\cdot} U_2 \tag{$\mathit{\Pi}_7$}$$

$$\theta \quad \leftarrow \quad T \underset{3(3E1+2S1)}{\cdot} U_1 \tag{$\mathit{\Theta}_7$}$$

For the supporting equations 9'–16' we have three cases.

- Equation 9' has variable vectors of length $l = 2$ ("borrowing" the variable $\underline{b_{1,i}}$ to increase the number of variables in $\mathbb{G}_1$ to 2 as well; this variable will have coefficient 0) and matrix $\mathit{\Gamma} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. We can apply this matrix to a vector of group elements "for free" since its effect is simply to zero out the second row. We can write the equations as

$$\pi \quad \leftarrow \quad R_1^\top \cdot \iota_2 \left(\boldsymbol{w} + \mathit{\Gamma} \cdot \boldsymbol{b} \right) + \left(R_1^\top \mathit{\Gamma} R_2 - T^\top \right) \cdot U_2 \tag{$\mathit{\Pi}_9$}$$

$$\theta \quad \leftarrow \quad R_2^\top \cdot \iota_1 \left(\boldsymbol{v} + \mathit{\Gamma} \cdot \boldsymbol{a} \right) + T \cdot U_1 \tag{$\mathit{\Theta}_9$}$$

  Like above, we compute the cost to get (for $l = 2$ and the extra l S_1 operations from $\boldsymbol{v} + \boldsymbol{a}$): $15E_1 + 15E_2 + 14S_1 + 14S_2$. This is the cost of a single equation; Equation 9' is as we have remarked earlier in fact a collection of L equations.
- Equations 10'–15' have vectors of length $l = 1$ and an all-zero matrix $\mathit{\Gamma}$ since no "quadratic" terms in the variables appear. The equations collapse to

$$\pi \quad \leftarrow \quad R_1^\top \underset{3E_2}{\cdot} \iota_2(\boldsymbol{w}) \underset{3S_2}{-} T^\top \underset{3(3E_2+2S_2)}{\cdot} U_2 \tag{$\mathit{\Pi}_{10}$}$$

$$\theta \quad \leftarrow \quad R_2^\top \underset{3E_1}{\cdot} \iota_1(\boldsymbol{v}) \underset{3S_1}{+} T \underset{3(3E_1+2S_1)}{\cdot} U_1 \tag{$\mathit{\Theta}_{10}$}$$

  for a total cost per individual equation of $12E_1 + 12E_2 + 9S_1 + 9S_2$.
- Equation 16' has three variables in $\mathbb{G}_2$ but none in $\mathbb{G}_1$. We can omit the cost of "inverting" π (the initial minus sign) since we could just invert the integers in the matrix T instead, which we assume is for free. The proof becomes

$$\pi \quad \leftarrow \quad -T^\top \underset{3(3E_2+2S_2)}{\cdot} U_2 \tag{$\mathit{\Pi}_{16}$}$$

$$\theta \quad \leftarrow \quad R_2^\top \underset{3(2E_1+S_1)}{\cdot} \iota_1(\boldsymbol{v}) \underset{3S_1}{+} T \underset{3(3E_1+2S_1)}{\cdot} U_1 \tag{$\mathit{\Theta}_{16}$}$$

  for a cost per equation of $15E_1 + 9E_2 + 12S_1 + 6S_2$.

C.4 Verification

Verifying a proof involves computing pairings. The size of the elements π and θ in a proof is fixed (3-vectors over the B modules for (S)DLIN) but the number of commitments to verify for each proof depends on the number of variables involved. Following Groth and Sahai [9], but using the corrected formula by Ghadafi et al. [25], we define a pairing operation on vectors of elements over the B-modules: for any integer l,

$$\otimes_l : {B_1}^l \times {B_2}^l \to B_T, \quad (\boldsymbol{u}, \boldsymbol{v}) \mapsto \prod_{i=1}^{l} e_B(u_i, v_i)$$

where the pairing $e_B : B_1 \times B_2 \to B_T$ over the plain B modules is defined in terms of the basic pairing e, viewing B-elements as vectors and matrices over the basic groups:

$$e_B\left((x_1, x_2, x_3), (y_1, y_2, y_3)\right) := \begin{pmatrix} e\left(x_1, y_1\right) & e\left(x_1, y_2\right) & e\left(x_1, y_3\right) \\ e\left(x_2, y_1\right) & e\left(x_2, y_2\right) & e\left(x_2, y_3\right) \\ e\left(x_3, y_1\right) & e\left(x_3, y_2\right) & e\left(x_3, y_3\right) \end{pmatrix}$$

The cost of an $\otimes_l$ operation is thus $9l \cdot P + 9(l-1)S_T$.

For the verification of a Groth-Sahai proof of a PPE of order (number of variables in each group) l with constants v, w and commitments to variables c, d for $\mathbb{G}_1$ and $\mathbb{G}_2$ variables respectively, the formula is

$$\iota_1(\boldsymbol{v}) \otimes_l \boldsymbol{d} \cdot \boldsymbol{c} \otimes_l \iota_2(\boldsymbol{w}) \cdot \boldsymbol{c} \otimes_l \mathit{\Gamma} \boldsymbol{d} \stackrel{?}{=} \iota_T(t) \cdot U_1 \otimes_3 \pi \cdot \theta \otimes_3 U_2 \tag{V}$$

For the equations in CKLM13, we can make several optimisations to reduce the number of pairing computations. The first transformation applies to all equations: we can use bilinearity and hence distributivity over the group operation to rewrite the second two pairings of vectors as $c \otimes_L (\iota_2(\boldsymbol{w}) \cdot \mathit{\Gamma}\boldsymbol{d})$ where the group operation $(\cdot)$ on vectors is to be understood component-wise. In some cases it is cheaper to factor out $\boldsymbol{d}$ instead, though.

- Equations 1'–6'. Here $l = L$ but $\boldsymbol{v} = \boldsymbol{0}$ and $\mathit{\Gamma}$ is the identity matrix so we are left with

$$\boldsymbol{c} \underset{\substack{9L\cdot P+\\ 9(L-1)S_T}}{\otimes_L} \left(\iota_2(\boldsymbol{w}) \underset{9LS_T}{\cdot} \boldsymbol{d}\right) \stackrel{?}{=} \iota_T(t) \underset{9S_T}{\cdot} U_1 \underset{\substack{27P+\\ 18S_T}}{\otimes_3} \pi \underset{9S_T}{\cdot} \theta \underset{\substack{27P+\\ 18S_T}}{\otimes_3} U_2 \tag{V_1}$$

- Equations 7', 8': $l = L$, $\boldsymbol{v} = \boldsymbol{0}$ and $\mathit{\Gamma}$ is all zeroes.

$$\boldsymbol{c} \underset{\substack{9L\cdot P+\\ 9(L-1)S_T}}{\otimes_L} \iota_2(\boldsymbol{w}) \stackrel{?}{=} \iota_T(t) \underset{9S_T}{\cdot} U_1 \underset{\substack{27P+\\ 18S_T}}{\otimes_3} \pi \underset{9S_T}{\cdot} \theta \underset{\substack{27P+\\ 18S_T}}{\otimes_3} U_2 \tag{V_7}$$

- Equation 9': $l = 2$, $\boldsymbol{w} = \boldsymbol{0}$ and applying $\mathit{\Gamma}$ is for free. We factor out $\boldsymbol{d}$:

$$\left(\iota_1(\boldsymbol{v}) \underset{18S_T}{\cdot} \boldsymbol{c}\right) \underset{18P+9S_T}{\otimes_2} \mathit{\Gamma}\boldsymbol{d} \stackrel{?}{=} \iota_T(t) \underset{9S_T}{\cdot} U_1 \underset{\substack{27P+\\ 18S_T}}{\otimes_3} \pi \underset{9S_T}{\cdot} \theta \underset{\substack{27P+\\ 18S_T}}{\otimes_3} U_2 \tag{V_9}$$

– Equations 10'–15': $l = 1$ and Γ is all zeroes; we are left with

$$\iota_1(\boldsymbol{v}) \underset{9P}{\otimes_1} \boldsymbol{d} \underset{9S_T}{\cdot} \boldsymbol{c} \underset{9P}{\otimes_1} \iota_2(\boldsymbol{w}) \overset{?}{=} \iota_T(t) \underset{9S_T}{\cdot} U_1 \underset{\substack{27P+\\18S_T}}{\otimes_3} \pi \underset{9S_T}{\cdot} \theta \underset{\substack{27P+\\18S_T}}{\otimes_3} U_2 \tag{V_{10}}$$

– Equation 16': $l = 3$ but $\boldsymbol{w} = \boldsymbol{0}$ and Γ is all zeroes:

$$\iota_1(\boldsymbol{v}) \underset{\substack{27P+\\18S_T}}{\otimes_3} \boldsymbol{d} \overset{?}{=} \iota_T(t) \underset{9S_T}{\cdot} U_1 \underset{\substack{27P+\\18S_T}}{\otimes_3} \pi \underset{9S_T}{\cdot} \theta \underset{\substack{27P+\\18S_T}}{\otimes_3} U_2 \tag{V}$$

C.5 Discussion

Our results indicate a rough upper bound on the cost of creating and verifying a CKLM13 proof. There are many optimisations which we have not yet considered and that could be analysed in future work, for example:

– Taking into account that 2 out of 3 group elements of any B-vector in the image of ι_j $(j \in \{1, 2\})$ are always zero.
– Taking into account that many of the constants in the supporting equations lie in $\{0, 1\}$, eliminating the need for further multiplications.
– Reusing intermediate results between the various equations rather than recomputing them from scratch each time. For example, the constants u_i, v'_i, w'_i appear twice each in eq. 1–6 and the variables $\underline{a_i}, \underline{b_i}$ even three times each, leading to possibilities for optimisation.
– Multi-exponentiation. Expressions of the form $\sum_{i=1}^{l} v_i X_i$ in a group $\mathbb{G}_j, j \in \{1, 2\}$ we counted as $l \cdot E_j + (l - 1)S_j$ yet algorithms exist to perform such multi-exponentiations[11] more efficiently than exponentiating each element individually and adding the results. Any expression of the form $R^\top \cdot \boldsymbol{b}$ or similar that looks like a "matrix-vector product" is in fact a multi-exponentiation and could be improved upon by such techniques.
– Applying batching techniques [34]; since CKLM13 proofs have large numbers of equations of very similar forms these should be readily applicable and in particular massively reduce the number of pairings required to verify a proof.

[11] Since we use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$, a group exponentiation is a *multiplication* in our notation.