# How to Verifiably Encrypt Many Bits for an Election?

Henri Devillez, Olivier Pereira, and Thomas Peters

UCLouvain – ICTEAM, B-1348 Louvain-la-Neuve – Belgium
{henri.devillez, olivier.pereira, thomas.peters}@uclouvain.be

**Abstract.** The verifiable encryption of bits is the main computational step that is needed to prepare ballots in many practical voting protocols. Its computational load can also be a practical bottleneck, preventing the deployment of some protocols or requiring the use of computing clusters. We investigate the question of producing many verifiably encrypted bits in an efficient and portable way, using as a baseline the protocol that is in use in essentially all modern voting systems and libraries supporting homomorphic voting, including ElectionGuard, a state-of-the-art open source voting SDK deployed in government elections. Combining fixed base exponentiation techniques and new encryption and ZK proof mechanisms, we obtain speed-ups by more than one order of magnitude against standard implementations. Our exploration requires balancing conflicting optimization strategies, and the use of asymptotically less efficient protocols that turn out to be very effective in practice. Several of our proposed improvements are now on the ElectionGuard roadmap.

## 1 Introduction

*Verifiable encryption of bits.* Encrypting bits is the main computational step that is needed in order to prepare a ballot in numerous voting protocols [5, 10–12] and systems, including VoteBox, Helios, STAR-Vote, Belenios, Strobe and ElectionGuard for instance [1, 3, 8, 4, 13, 22]. In these protocols, which follow the general approach pioneered by Benaloh [5], voters compute one additively homomorphic ciphertext per candidate on the ballot, and prove (in zero-knowledge) that each of these ciphertexts encrypts a bit, expressing whether the voter supports the candidate or not. Thanks to the homomorphic property, the ciphertexts submitted by the voters can be combined candidate-wise, resulting in a vector of ciphertexts encrypting the number of votes that each candidate received, and these ciphertexts can then be verifiably decrypted in order to obtain the election result.

The blueprint that we just described is adequate for approval voting, where voters are allowed to support as many candidates as they want. Different ballot completion rules may require additional verifiable bit encryptions. For example, if voters can support at most $k$ candidates out of $n$, one common approach to proving the validity of ballots, based on the addition of dummy candidates, requires to compute the verifiable encryption of $k + n$ bits [11, 13]. For Instant-Runoff voting (IRV), some protocols require to produce a number of verifiably encrypted bits that is equal to the square of the number of candidates [21].

Apart from voting applications, which are our main focus here, the verifiable encryption of bits is also a central component of other protocols. A prominent example is the computation of range proofs: there, one of the standard approaches to prove that a ciphertext encrypts a value $v$ less than $2^n$ consists in producing $n$ verifiable encryptions of each of the bits of $v$ [2], and arbitrary ranges can be supported with $2n$ verifiable bit encryptions [23].

*The computational cost of encrypting bits.* All recent voting protocol with homomorphic tallying implementations encrypt bits using exponential ElGamal, that is, a bit $v$ is encrypted as a pair $(g^r, g^v h^r)$, and the proof that a ciphertext encrypts a bit is computed as a disjunctive Chaum-Pedersen proof [7, 9].

The choice of ElGamal over Paillier encryption and its variants [11, 12] is motivated by the simplicity to generate keys for a distributed or threshold variant of ElGamal [19], compared to the challenges of generating an RSA modulus in a distributed way [14]. The disjunctive Chaum-Pedersen proof was adopted and used in virtually every system since the initial proposal by Cramer et al. [10]. In this approach, the cost of verifiably encrypting a bit is largely dominated by 7 modular exponentiations: 2 for the ElGamal encryption, and 5 for the proof. This cost can quickly become limitating in practice.

Let us consider, as a motivating example, the state-of-the-art ElectionGuard SDK developed by Microsoft, which has been used in various public elections since 2020 [13]. When ElectionGuard was deployed for a Risk Limiting Audit in Inyo County, the encryption of a single ballot took around 6 seconds, the exponentiations being reported as the bottleneck [4]. In the context of such an audit, thousands and possibly millions of ballots have to be verifiably encrypted, which led to the deployment of a cluster, raising numerous practical challenges and requiring a significant expertise [24].

As a second example, we can turn to the Verificatum JavaScript Cryptographic Library [25], which is a state-of-the-art crypto library supporting various ElGamal-related operations. A library benchmark shows that a modular exponentiation computed in the group used in ElectionGuard takes around 37ms. on a 2020 laptop. Encrypting a ballot with 100 candidates, a size that is typical in many countries, would then require around 26 sec.. This may create important usability issues on a laptop, and would just be unbearable on a slow lower-end smartphone.

## 1.1 Contributions

Taking the state-of-the art ElectionGuard SDK as our baseline, we show how to considerably improve the speed at which bits can be verifiably encrypted, with a focus on the constraints from voting applications.

We proceed in 4 steps:

1. We observe that verifiable bit encryption can take advantage of fixed base encryption techniques, something that is not accounted for in existing libraries.

Taking into account that fairly large amounts of memory (at least a few MB) are available on voting devices, we depart from standard techniques that focus on memory constrained environments and explore the use of a fast and memory-intensive approach.
2. We explore the use of multi-ElGamal encryption, that reduces the number of exponentiations at the cost of requiring a larger number of bases. We show that this approach offer benefits when the precomputation time does not need to be accounted for when preparing a ballot, or when the number of public keys remains relatively low.
3. We propose a switch from the traditional disjunctive Chaum-Pedersen proofs to product proofs. We show that, within the space of protocol design strategies that keep a linear number of multiplication operations (in the number of encrypted bits), the proof computational effort can be halved for multi-ElGamal ciphertexts, and an extra halving can be obtained when many proofs need to be computed, taking the cost of the proof close to 1 exponentiation per encrypted bit. .
4. Eventually, we explore the impact of recent developments that aim at providing short proofs. The use of these techniques is intriguing because the proof size is not our primary goal, and the number of multiplications required to compute these proofs is typically super-linear (typically in $\mathcal{O}(n \log^d(n))$ with $d \geq 1$), compared to the linear cost of the Chaum-Pedersen and product proofs. Nevertheless, we propose a new protocol that offers speed improvements for any practical number of proofs to be computed, including by a factor up to 70 for a few thousand proofs.

We benchmark our solutions against our baseline protocol and confirm their benefits. Several of our improvements are now on the ElectionGuard 2.0 roadmap.

## 2 The Baseline

In all the voting systems based on homomorphic tallying that we examined, including [1, 22, 8, 13], the verifiable encryption of bits is performed as in the original protocol of Cramer et al. [10], which can be described as follows.

1. A group $\mathbb{G}$ is chosen as a subgroup of prime order $q$ of a group $\mathbb{Z}_p^*$, with $|q| = 256$ and $2048 \leq |p| \leq 4096$ depending on the implementation. A generator $g$ of $\mathbb{G}$ is also chosen. Then, an ElGamal public key $h \in \mathbb{G}$ is selected, with the corresponding secret key $x : h = g^x$ being kept secret by a group of trustees. (Distributed key generation protocols are used for that.)
2. A bit $v$ is encrypted as an exponential ElGamal pair $(g^r, g^v h^r)$ for a random $r \leftarrow \mathbb{Z}_q$.
3. A disjunctive Chaum-Pedersen proof [7, 9] is computed in order to demonstrate that $v \in \{0, 1\}$. Given a ciphertext $(A, B) = (g^r, g^v h^r)$ as above, a commitment is computed as a pair of ciphertexts $(A'_0, B'_0) = (g^s, g^{vw} h^s)$ and $(A'_1, B'_1) = (g^t, g^{(1-v)w} h^t)$ where $s, t, w \leftarrow\$ \mathbb{Z}_q$. Then a random challenge $e \in \mathbb{Z}_q$ is obtained using the Fiat-Shamir transform. Eventually, the

sub-challenges $e_0 = (1-v)e - w$ and $e_1 = ve + w$ are computed, as well as the responses $f_0 = s + e_0 r$ and $f_1 = t + e_1 r$. The proof is made of $(A'_0, B'_0, A'_1, B'_1, e_0, e_1, f_0, f_1)$.

In real-world implementations, the choice of a subgroup of $\mathbb{Z}_p^*$ is preferred over a group on elliptic curves, motivated by the desire to keep the implementation of a verifier as accessible as possible: the basics of modular arithmetic are part of any CS curriculum, which is not the case of elliptic curves. Exponential ElGamal is preferred over Paillier, which is also additively homomorphic and has a more efficient decryption process, because ElGamal comes with efficient threshold key generation protocols. The disjunctive Chaum-Pedersen proof can be described in various ways. Here, we follow ElectionGuard, whose specification can be accessed for further details [13]. Numerous other descriptions exist and lead to equally or less efficient implementations.

We observe that the verifiable encryption of a bit requires 2 exponentiations for the ciphertext and 5 more exponentiations for the proof. In terms of storage, the ciphertext takes 2 elements in $\mathbb{Z}_p^*$, and the proof takes 4 elements in $\mathbb{Z}_p^*$ and 4 elements in $\mathbb{Z}_q$. Using the ElectionGuard parameters with $|p| = 4096$, we see that each verifiably encrypted bit requires 25600 bits $\approx$ 3KB. However (and even though this is usually not the case in practice), the proof size can be much reduced by omitting the 4 elements in $\mathbb{Z}_p^*$, which can be recomputed from the other ones, taking the size down to about 1KB.

## 3 Fixed-base exponentiation

Existing implementations of homomorphic voting schemes (e.g., VoteBox, Helios, Belenios, ElectionGuard...) make use of the exponentiation function of standard Multi-Precision arithmetic libraries for computing modular exponentiations, including gmpy2 [15] in Python and jsbn [26] in JavaScript. These libraries support the computation of modular exponentiations as a stateless operation.

Numerous techniques however exist that make it possible to compute multiple exponentiations w.r.t. a single base much faster than independently [6, 18]. This is precisely our case here: we only use bases $g$ and $h$.

The design of most fixed base exponentiation algorithms was however guided by constraints that are quite different of those of voting exponentiations: while these algorithms behave very well when a small amount of memory is available to store the result of precomputation, voting applications can typically dedicate several MB, and possibly even GB of memory to precomputation storage.

It is tempting to consider such an option, given that we may need to compute a lot of exponentiations: a single race with half a dozen candidates will already require a few dozens exponentiation, and a full ballot, which can often contain one or two hundreds choices, can take a thousand exponentiations. Even more challenging is the encryption of all the ballots cast in an election, as needed for a privacy preserving publicly verifiable risk limiting audit, which may require millions of exponentiations. Based on these observations, we explore the use of a simple precomputation approach based on the standard right-to-left $k$-ary exponentiation algorithm, aiming at minimizing the number of multiplications needed per exponentiation. We will compare it to other traditional approaches below.

*Precomputation* Suppose that we are willing to compute a lot of exponentiations in base $g$, with exponents of at most $\ell$ bits. We select a parameter $k$, and precompute a table of $t = \lceil \ell/k \rceil$ lines and $2^k$ columns, in which $table[i][j] \leftarrow g^{2^i \cdot j}$. Such a table can be computed using $t \cdot (2^k - 1)$ multiplications as the first column of "1"'s, for $j = 0$, requires no computation. As an example, for $\ell = 6$ and $k = 2$, the table looks as follows:

| | | | |
|---|---|---|---|
| 1 | $g$ | $g^2$ | $g^3$ |
| 1 | $g^4$ | $g^8$ | $g^{12}$ |
| 1 | $g^{16}$ | $g^{32}$ | $g^{48}$ |

*Computation* Computing $g^e$ for $e = (e_{t-1} \ldots e_0)_{2^k}$ with $e_i \in \{0, \ldots, 2^k - 1\}$ is now immediate: we just need to pick the correct element on each line of the table, and multiply them together: $g^e = \prod_{i=0}^{t-1} table[i][e_i]$.

This algorithm requires $t-1$ multiplications, and makes use of the first column of the table in order to deal with the cases where some $e_i = 0$. An alternative would be to simply exclude these terms from the product but, since we intend to use relatively large values of $k$, this strategy would only save us a marginal amount of memory and computation, while adding a test on each $e_i$ value.

*How to choose k?* We see that, for a fixed exponent size, the number of multiplications and the storage that are required for the precomputation grow like $2^k/k$, while the online computation decreases like $1/k$.

Obviously, if the precomputation time does not matter (because it can be performed well in advance), choosing a value of $k$ as large as the memory can fit would lead to the fastest online exponentiations. If we would like to minimize the total computation time, then the right balance needs to be found between the time spent on precomputation and the time spent on computation: for $n$ exponentiations in base $g$, the total number of multiplications is $\lceil \ell/k \rceil (2^k - 1) + (\lceil \ell/k \rceil - 1)n$. When $n \gg 1$, this expression is minimum when $n \approx (\ln(2)k - 1)2^k$.

For the sake of concreteness, we explore these values in the group used in ElectionGuard, that is $\ell = |q| = 256$. Table 1 contains the maximum value of $n$ until which various choices of $k$ are optimal, based on the multiplication count made above. We may observe that $k = 15$ and $k = 17$ are never optimal choices.

The storage that is required for the precomputation table grows relatively fast: if we ignore the first column that only contains "1", we need to store $t \cdot (2^k - 1)$ group elements. Table 1 also shows these volumes for various values of $k$. Even if this grows fast, the volumes remain lower than 100MB for values of $k$ up to 13, which should be within reach of any modern computer. The table also shows that the benefits of increasing the value of $k$ for such values also starts plummeting: moving from $k = 7$ to $k = 10$ gives a speedup by a factor 36/25=1.44 for an

| $k$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| $n$ | 16 | 54 | 121 | 332 | 692 | 2219 | 3926 |
| Table size (MB) | 0.3 | 0.5 | 0.8 | 1.4 | 2.4 | 4.2 | 7.6 |
| $t-1$ (online mult.) | 85 | 63 | 51 | 42 | 36 | 31 | 28 |
| $k$ | 10 | 11 | 12 | 13 | 14 | 16 | 18 |
| $n$ | 11265 | 20481 | 36865 | 147457 | 245761 | 2883585 | 3407873 |
| table size (MB) | 14 | 25 | 46 | 84 | 159 | 537 | 2013 |
| $t-1$ (online mult.) | 25 | 23 | 21 | 19 | 18 | 15 | 14 |

Table 1: The $n$ line gives an estimation of the maximum value of $n$ for which choices of $k$ are optimal. For instance, $k = 8$ is the best choice for $n \in [693, 2219]$. We also give the precomputation storage volume and the online computation work for various parameters of $k$.

extra 11.6MB of storage, while moving from $k = 10$ to $k = 13$ only gives an extra factor 25/19=1.31 for an extra 70MB of storage.

Overall, we observe a few "sweet spots" in this table: in a low-memory setting, we see that computational gains remain fairly high until we reach $k = 8$ and $t - 1 = 31$, which still comes with a very small memory requirement of 4MB. The value $k = 13$ and $t - 1 = 19$ is the second-to-last that saves at least 2 online multiplications compared to the previous value of $k$ and keeps memory requirements below 100MB. As we will see in our benchmarks in Section 7.1, all these values offer important speed improvements over the standard exponentiation function of big integer libraries.

*What about other fixed-base exponentiation methods?* The method that is described above is demanding in terms of precomputation table size, compared to traditional solutions. Nevertheless, we see that, when aiming for very fast exponentiations, it is quite competitive, and remarkably simple. To offer some points of comparison, we consider the classical methods as described in [18] for the parameters listed above.

– The fixed-base windowing method (Algo. 3.41) is expected to require $2^k + t - 3$ multiplications/exponentiation and the storage of $t$ values. This is minimum for $k = 4$ in our case, and requires 77 multiplications per exponentiation and the storage of 63 group elements. So, our method leads to faster exponentiations as soon as $k > 4$.
– The fixed-base comb method (Algo. 3.44) is expected to require $2t - 2$ multiplications per exponentiation, and the storage of $2^k$ precomputed values. If we aim for the same number of multiplications/exponentiation (meaning that $k$ needs to be approximately twice as big for the comb method compared to our method), our method is more efficient as soon as $k > 5$.
– The two-table fixed-base comb method (Algo 3.45) is expected to require $3t/2 - 2$ multiplications/exponentiation, and the storage of $2^{k+1}$ precomputed values. Here, our method is more efficient as soon as $k > 7$.

So, it seems that, apart from its extreme simplicity, the method we described also offers important speed-ups. Our estimates only focus on the number of multiplications, being the bulk of the work here. More sophisticated methods focusing on the efficient computation of short multiplication chains, for example Pippenger's [20], may require a smaller number of multiplications. However, they also require more bit-by-bit inspection in the exponents, and create multiplication chains that combine all the exponents, which requires additional book keeping.

## 4 Multi-ElGamal

The implementation improvement discussed above does not touch the voting protocol itself, easing its integration in an existing system. Nevertheless, it is appealing to explore whether the use of other cryptographic mechanisms would reduce the efforts needed to verifiably encrypt a bit. We start by exploring the case of ElGamal encryption, and will turn to the ZK proof in the next sections.

ElGamal encryption requires 2 exponentiations per bit. But ElGamal encryptions can be easily batched if we have multiple public keys $(h_1, \ldots, h_m) = (g^{x_1}, \ldots, g^{x_m})$ with each $x_i \leftarrow \mathbb{Z}_q$: we can encrypt $m$ bits $(v_0, \ldots, v_m)$ as $(g^r, g^{v_1} h_1^r, \ldots, g^{v_m} h_m^r)$ – the security of this multi-ElGamal scheme can be reduced to the one of the original ElGamal encryption scheme. We can now encrypt $m$ bits with $m+1$ exponentiations, compared to $2m$ with plain ElGamal, leading to a speed-up by a factor close to 2 even for relatively low values of $m$.

However, exponentiations are now computed w.r.t. $m+1$ bases instead of 2, which may require more efforts of precomputation if we want to use fixed-base exponentiation methods. If the precomputation is taken offline, and in the absence of memory concerns, multi-ElGamal will always be more efficient.

But if the precomputation needs to be made online, then what we gain on one side may be lost on the other side. We can estimate this by exploring a few values by multiplication counts. If $n = 1000$, the two exponentiations of plain ElGamal (i.e., $m = 1$) require an optimal effort of 78320 multiplications for $k = 8$ (including precomputation), and a storage of 8MB. The use of multi-ElGamal can offer some benefits: we reach 66045 multiplications for $m = 4$ and $k = 6$, and a slightly lower precomputation volume of 7MB. If $n = 100000$, plain ElGamal requires an optimal effort of 4.13 million multiplications for $k = 13$. If we switch to multi-ElGamal, we can for instance obtain 3.06 millions multiplications for $m = 5$ and $k = 11$. The storage needed for the tables again slightly decreases from 168MB to 151MB. Overall, we observe that the benefits increase when we have more votes to encrypt. But they remain well below the factor $\approx 2$ that was hoped for a large $m$.

The adoption of multi-ElGamal may also be complicated by extra validity requirements on ballots. It is for instance quite common to require that a maximum number of candidates are selected within a single race. This is typically handled by computing an encryption of the number of candidates selected within the race as the homomorphic sum of the ciphertexts computed for each choice, and proving that this sum is within the expected range. However, this homomorphic addition won't work if the choices within a race are encrypted with a different public key $h_i$. Nevertheless, it remains an option to use multi-ElGamal in an election with multiple races, and to use one public key $h_i$ per race.

## 5 Adapting the ZK 0-1 proofs – Linear Techniques

The disjunctive version of the Chaum-Pedersen protocol described in Section 2, requires 5 modular exponentiations: 3 in base $g$, and 2 in base $h$, and makes most of the computational effort.

We will first see how a simple change in the ElGamal encryption process makes it possible to save 1 exponentiation in base $g$, down to a total of 4 exponentiations. As a second step, we will turn to proofs for multi-ElGamal ciphertexts, and show how to compute the proof with $3m+1$ exponentiations for an $m$-key multi-ElGamal ciphertext. Eventually, using batching techniques, we will show how to compute a proof for $\ell$ multi-ElGamal ciphertexts with $\ell \cdot m + \ell + m + 1$ exponentiations, bringing the cost of the proof down to almost 1 exponentiation per encrypted bit.

### 5.1 From 5 to 4 Exponentiations

Looking back at the disjunctive Chaum-Pedersen proof as it is described in Section 2, we can observe that the computation of $(A'_0, B'_0) = (g^s, g^{vw}h^s)$ and $(A'_1, B'_1) = (g^t, g^{(1-v)w}h^t)$ requires a total of 3 exponentiations when $v$ is 0 or 1.

We observe that exponential ElGamal encryption works just as well, and may be slightly more efficient by saving one multiplication, if bits are encrypted as a $(g^r, h^{v+r})$ pair. Now, the commitment of the proof can be computed with a pair of ciphertexts $(A'_0, B'_0) = (g^s, h^{vw+s})$ and $(A'_1, B'_1) = (g^t, h^{(1-v)w+t})$, and the rest of the proof can remain unchanged. This saves 1 exponentiation in base $g$, taking the cost of computing a proof from 5 to 4 exponentiations.

### 5.2 A 0-1 product proof

The adaptation of the disjunctive Chaum-Pedersen proof to multi-ElGamal ciphertexts does not offer any particular benefit, unfortunately: one basically needs to compute one full proof for each $(A, B_j) = (g^r, h_j^{v_j+r})$ pair, keeping a cost of 4 exponentiations per encrypted bit.

In Table 2, we describe another proof approach, that consists in proving that $v_j(1-v_j) = 0$, adapting a classical approach described in [17] for instance, and see that it makes it possible to take the cost of the 0-1 proof for a multi-ElGamal ciphertext down to $3m+1$ exponentiations.

### 5.3 Batching the product proof

Another advantage of the product proof is that it becomes compatible with batching techniques. To compress the proof further and save the computation of some exponentiations, and even more when we have $\ell$ multi-ElGamal ciphertexts,

**Commitment** The prover computes: $A' = g^s$, $B'_j = h_j^{w_j+s}$, $C_j = g^{t_j}$, $D_j = {h_j}^{w_j v_j + t_j}$, where $s, w_j, t_j \leftarrow\$\, \mathbb{Z}_q$ and $j \in [m]$.
**Challenge** The verifier sends a challenge $e \leftarrow\$\, \mathbb{Z}_q$.
**Response** The prover computes the response, for $j \in [m]$:

$$f_r = s + er\,, \qquad f_{v_j} = w_j + ev_j\,, \qquad f_{u_j} = t_j + r(e - f_{v_j})\,.$$

**Verification** The verification proceeds by checking that, for every $j \in [m]$:

$$A^e A' = g^{f_r}\,, \quad B_j^e \bar{B}'_j = h_j^{f_{v_j}+f_r}\,, \quad A^{e-f_{v_j}} C_j = g^{f_{u_j}}\,, \quad B_j^{e-f_{v_j}} D_j = h_j^{f_{u_j}}\,.$$

Table 2: Proof of 0-1 encryption for multi-ElGamal ciphertexts.

we consider a batching process. More precisely, when we have $\ell$ ciphertexts of the form $(A_i, \{B_{ij}\}_{j=1}^m) = (g^{r_i}, \{h_j^{v_{ij}+r_i}\}_{j=1}^m)$, for $i \in [\ell]$, we seek to prove that $\sum_{ij} v_{ij}(1 - v_{ij}) \cdot \alpha^{(i-1)m+j-1} = 0$, where $\alpha \in \mathbb{Z}_q$ is a random value. As long as $\alpha$ is independent of the statements to be proven, the Schwartz-Zippel lemma implies that the above equation (seen as the evaluation of a polynomial at the random point $\alpha$) ensures $v_{ij} \in \{0, 1\}$, for all $i \in [\ell]$ and $j \in [m]$, with overwhelming probability $1 - \ell m/q$.

Our protocol is in Table 3. In the commit phase, we provide the $A'$ and $B'$ elements from which we can extract all the $r_i$ and $v_{ij}$ exponents, as in our previous protocol. This makes it possible to isolate all these exponents from each other before starting the batching.

Now, the batching proceeds by picking a random exponent $\alpha$ after all the ciphertexts have been chosen and compressing the $C$ and $D$ terms of our previous proof: using $\alpha$, the $\ell m$ $C_{ij}$ terms can be compressed into a single group element $C_0$ and the $\ell m$ $D_{ij}$ terms can be compressed into $m$ group elements, one per $h_j$ base. With respect to $\ell$ parallel executions of the first protocol, we would have the relation $C_0 = \prod_{i\in[\ell],j\in[m]} C_{ij}^{\alpha^{(i-1)m+j-1}}$ and $D_{0j} = \prod_{i\in[\ell]} D_{ij}^{\alpha^{(i-1)m+j-1}}$.

It would be tempting to further compress our $D_{0j}$'s into a single $D_0$ as we do for $C_0$, but the special-soundness would then have to rely on the hardness of computing the discrete logarithms of the $h_j$'s in basis $g$. While the security of the encryption implies this hardness, the authorities that know the secret key would have the possibility to cheat when colliding with corrupted users/provers. This is the reason why we keep our $m$ values.

We prove the special-soundness of our protocol below – the other standard properties of $\Sigma$-protocols come by inspection.

**Theorem 1.** *The protocol in Table 3 has special soundness.*

*Proof.* From any two transcripts of this protocol with identical $\alpha$ and commitments $(A'_i, \{B'_{ij}\}_{j=1}^m)_{i=1}^\ell, C_0, \{D_{0j}\}_{j=1}^m$ different challenges $e$ and $e'$ and responses $(f_{r_i}, \{f_{v_{ij}}\}_{j=1}^m, \{f_{u_{ij}}\}_{j=1}^m)_{i=1}^\ell$ and $(f'_{r_i}, \{f'_{v_{ij}}\}_{j=1}^m, \{f'_{u_{ij}}\}_{j=1}^m)_{i=1}^\ell$, we can extract, for all $i \in [\ell]$ and $j \in [m]$,

$$r_i = \frac{f_{r_i} - f'_{r_i}}{e - e'}\,, \qquad v_{ij} = \frac{f_{v_{ij}} - f'_{v_{ij}}}{e - e'}\,, \qquad u_{ij} = \frac{f_{u_{ij}} - f'_{u_{ij}}}{e - e'}\,,$$

**Statement** Given the statement $A_i = g^{r_i}$, $\{B_{ij} = h_j^{v_{ij}+r_i}\}_{j=1}^m$, for $i \in [\ell]$, the verifier generates and sends $\alpha \leftarrow\$\, \mathbb{Z}_q$ to the prover.

**Commitment** The prover computes $A'_i = g^{s_i}$, $B'_{ij} = h_j^{w_{ij}+s_i}$ for proving openings,

$$C_0 = g^{\sum_{i\in[\ell],j\in[m]} t_{ij}\cdot\alpha^{(i-1)m+j-1}}\,, \qquad D_{0j} = h_j^{\sum_{i\in[\ell]}(v_{ij}w_{ij}+t_{ij})\cdot\alpha^{(i-1)m+j-1}}$$

for proving the relations, where $s_i, w_{ij}, t_{ij} \leftarrow\$\, \mathbb{Z}_q$, for all $i \in [n], j \in [m]$.

**Challenge** The verifier sends a challenge $e \leftarrow\$\, \mathbb{Z}_q$.

**Response** The prover computes the response for all $i \in [\ell], j \in [m]$:

$$f_{r_i} = er_i + s_i\,, \qquad f_{v_{ij}} = ev_{ij} + w_{ij}\,, \qquad f_{u_{ij}} = r_i(e - f_{v_{ij}}) + t_{ij}\,.$$

**Verification** The verification proceeds by checking that, for all $i \in [\ell], j \in [m]$:

$$A_i^e A'_i = g^{f_{r_i}}\,, \qquad B_{ij}^e B'_{ij} = h_j^{f_{v_{ij}}+f_{r_i}}\,,$$

$$\prod\nolimits_{i\in[n],j\in[m]} A_i^{(e-f_{v_{ij}})\cdot\alpha^{(i-1)m+j-1}} C_0 \;=\; g^{\sum_{i\in[n],j\in[m]} f_{u_{ij}}\cdot\alpha^{(i-1)m+j-1}}\,,$$

$$\prod\nolimits_{i\in[n]} B_{ij}^{(e-f_{v_{ij}})\cdot\alpha^{(i-1)m+j-1}} D_{0j} \;=\; y_j^{\sum_{i\in[n]} f_{u_{ij}}\cdot\alpha^{(i-1)m+j-1}}\,.$$

Table 3: Batch proof of 0-1 encryption for multi-ElGamal ciphertexts.

where $(r_i, \{v_{ij}\}_{j=1}^m)_{i=1}^\ell$ are the exponents of the ciphertexts, since dividing the first two verification equations gives $A_i^{e-e'} = g^{f_{r_i}-f'_{r_i}}$ and $B_{ij}^{e-e'} = h_j^{(f_{v_{ij}}-f'_{v_{ij}})+(f_{r_i}-f'_{r_i})}$, and the encryption scheme is perfectly binding.

It remains to show that $v_{ij} \in \{0,1\}$, for $i \in [\ell], j \in [m]$. If we divide the remaining verification equations of the two transcripts by corresponding equations, and raise them all to the power $(e-e')^{-1} \bmod q$, then take the discrete logarithms in their respective basis, we get:

$$\sum\nolimits_{i\in[\ell],j\in[m]} r_i(1-v_{ij})\cdot\alpha^{(i-1)\ell+j-1} = \sum\nolimits_{i\in[\ell],j\in[m]} u_{ij}\cdot\alpha^{(i-1)\ell+j-1}\,,$$

$$\sum\nolimits_{i\in[\ell]} (v_{ij}+r_i)(1-v_{ij})\cdot\alpha^{(i-1)\ell+j-1} = \sum\nolimits_{i\in[\ell]} u_{ij}\cdot\alpha^{(i-1)\ell+j-1}\,,$$

for all $j \in [m]$. By injecting the values of the right-hand side member of the last equations for $j \in [m]$ into the one above we find

$$\sum_{i\in[\ell],j\in[m]} r_i(1-v_{ij})\cdot\alpha^{(i-1)\ell+j-1} = \sum_{i\in[\ell],j\in[m]} (v_{ij}+r_i)(1-v_{ij})\cdot\alpha^{(i-1)\ell+j-1}\,,$$

where the constants in front of the powers of $\alpha$ are uniquely determined by the statement. Since $\alpha$ was generated after the verifier received the statement, we have $r_i(1-v_{ij}) = (v_{ij}+r_i)(1-v_{ij})$, for all $i \in [\ell], j \in [m]$, due to the Schwartz-Zippel lemma, which implies that $0 = v_{ij}(1-v_{ij}) \bmod q$. $\square$

*Efficiency.* Computing the proof in Table 3 requires $(\ell+1)(m+1)$ exponentiations. The number of multiplications in the exponents increases to $4\ell m + \ell$ multiplications, but this will remain negligible in the group we consider where $|p| = 16|q|$. We can make a few more observations:

– If we need to compute $n = \ell \cdot m$ 0-1 proofs and if there is no precomputation, then picking $m \approx \ell$ is the best choice. In this case, when $n$ is large, the cost of the proof comes close to 1 exponentiation per encrypted bit.

- However, when precomputation is used, and since the number of exponentiations in each base is well-balanced, the remarks made for the choice of $m$ in the multi-ElGamal encryption still apply: we can expect only marginal benefits when increasing $m$ above 4, for most values of $n$.
- While our basic product proof did not offer any benefit for a regular ElGamal ciphertext ($m = 1$), the batching process helps quite a bit in that case: we move from $4n$ exponentiations down to $2n + 2$.

## 6 Adapting the ZK 0-1 proofs - Logarithmic batching

We propose a shorter proof system showing that $n = 2^\tau$ ElGamal ciphertexts encrypt bits. Our system shares similarities with a protocol due to Groth [16] that was described recursively as a subroutine of a bigger protocol. Here, we give an iterative description which *halves* the number of rounds of [16] to prove that $n$ pairs of commitments $\{(c_i, d_i)\}_{i=0}^{n-1}$, for any homomorphic commitment $\mathsf{Com}$ (ElGamal encryption in our case), satisfy $\sum_{i=0}^{n-1} x_i y_i = 0$, where $c_i = \mathsf{Com}(x_i; r_i)$ and $d_i = \mathsf{Com}(y_i; s_i)$ for some coins $r_i, s_i \in \mathbb{Z}_q$, for all $i = 0, \dots, n-1$. That is, we prove that the inner product is null. We then show how to turn our protocol into our desired proof system and analyze the efficiency.

### 6.1 Notations

We identify the index $i = \sum_{k=1}^{\tau} i_k 2^{k-1}$ with the $\tau$-bit string multi-index $i_1 \dots i_\tau$ so that $x_i = x_{i_1 \cdots i_\tau}$, for all $0 \le i \le 2^\tau - 1$, and conversely for all $0 \le i_1, \dots, i_\tau \le 1$. To get a shortened form of $x_{i_1 \cdots i_{k-1} i_k i_{k+1} \cdots i_\tau}$, we write $x_{i_k^- i_k i_k^+}$ with $i_k^- = i_1 \cdots i_{k-1}$ and $i_k^+ = i_{k+1} \cdots i_\tau$ when $\tau$ is implicit. By convention $i_1^-$ and $i_\tau^+$ are empty strings so that we have $x_{i_1^- i_1 i_1^+} = x_{i_1 i_1^+}$ as well as $x_{i_\tau^- i_\tau i_\tau^+} = x_{i_\tau^- i_\tau}$. In the same spirit, we set $x_{i_0^+} = x_{i_1 \cdots i_\tau}$. For the sake of readability, we often do not specify the values taken by the multi-indexes in the summation

$$\sum_{i_k^- i_k i_k^+} = \sum_{i_k^- \in \{0,1\}^{k-1}} \sum_{i_k \in \{0,1\}} \sum_{i_k^+ \in \{0,1\}^{\tau-k}}$$

since the bit-strings take all their possible values determined by the bit-size.

Similarly for exponents, we use a notation to compress the product $\alpha_1^{i_1} \cdots \alpha_\tau^{i_\tau}$ over $\mathbb{Z}_q$ as $\alpha^I$ when $\alpha = (\alpha_1, \dots, \alpha_\tau)$ and $I = i_1 \cdots i_\tau$. For the product of the first $k-1$ factors, with $\alpha_{<k} = (\alpha_1, \dots, \alpha_{k-1})$, we naturally write $\alpha_{<k}^{i_k^-} = \alpha_1^{i_1} \cdots \alpha_{k-1}^{i_{k-1}}$. By convention $\alpha_{<1} = ()$ and $\alpha_{<1}^{i_1^-} = 1$. We also see $k$-bit strings as tuples of $\mathbb{Z}^k$ so that, component-wise, $i_k^- - j_k^-$ is well defined. Finally, $x_I = x_{i_1 \cdots i_\tau}$.

### 6.2 Intuition

Assuming that the prover receives $\tau \in \mathcal{O}(\log n)$ unpredictable scalars $\alpha_1, \dots, \alpha_\tau$ of $\mathbb{Z}_q$ from the verifier, both parties can efficiently compute

$$C = \prod_{i_1, \dots, i_\tau = 0}^{1} c_{i_1 \dots i_\tau}^{\alpha_1^{i_1} \cdots \alpha_\tau^{i_\tau}} = \prod_{I \in \{0,1\}^\tau} c_I^{\alpha^I} = \mathsf{Com}\Big(\sum_I x_I \alpha^I; \sum_I r_I \alpha^I\Big)$$

as well as

$$D = \prod_{i_1,\dots,i_\tau=0}^{1} d_{i_1\dots i_\tau}^{\alpha_1^{-i_1}\cdots\alpha_n^{-i_\tau}} = \mathsf{Com}\Big(\textstyle\sum_I y_I\alpha^{-I}; \sum_I s_I\alpha^{-I}\Big) \ .$$

Letting zero-knowledge apart for now, the prover can send the opening $(X, R)$ of $C = \mathsf{Com}(X, R)$ to the verifier so that both parties can also compute the commitment $D^X = \mathsf{Com}(Y; S)^X = \mathsf{Com}(XY; XS) = \mathsf{Com}(Z, T)$ where $Z = XY = \sum_{I,J} x_I y_J \alpha^{I-J}$.

Viewed as a rational fraction over $\mathbb{Z}_q(\alpha)$ with $\tau$ indeterminates $\alpha = (\alpha_1, \dots, \alpha_\tau)$, the prover has to ensure that the constant term of $Z(\alpha)$ is $\sum_I x_I y_I = 0$. To get a proof of size in $\mathcal{O}(\tau)$ we rely on the following observation:

$$Z(\alpha) = \underbrace{\left(\sum_{i_\tau^-, j_\tau^-} x_{i_\tau^- 0} \cdot y_{j_k^- 1} \cdot \alpha_{<\tau}^{i_\tau^- - j_\tau^-}\right)}_{U_{\tau-1}} \cdot \alpha_\tau^{-1}$$
$$+ \underbrace{\sum_{i_\tau^-, j_\tau^-, i_\tau} x_{i_\tau^- i_\tau} \cdot y_{j_k^- i_\tau} \cdot \alpha_{<\tau}^{i_\tau^- - j_\tau^-}}_{V_{\tau-1}} + \underbrace{\left(\sum_{i_\tau^-, j_\tau^-} x_{i_\tau^- 1} \cdot y_{j_k^- 0} \cdot \alpha_{<\tau}^{i_\tau^- - j_\tau^-}\right)}_{W_{\tau-1}} \cdot \alpha_\tau \ ,$$

where the terms $U_{\tau-1}, V_{\tau-1}, W_{\tau-1}$ no more depend on $\alpha_\tau$. By iterating this process with the middle term we find

$$Z(\alpha) = U_{\tau-1}\alpha_\tau^{-1} + \cdots + U_0\alpha_1^{-1} + V_0 + W_0\alpha_1 + \cdots + W_{\tau-1}\alpha_\tau \ ,$$

for $V_0 = \sum_I x_I y_I$ and some $U_0, W_0, \dots, U_{\tau-1}, W_{\tau-1} \in \mathbb{Z}_q(\alpha)$, where for each $k = 1, \dots, \tau$, the terms $U_{k-1}, W_{k-1}$ only depends on $\alpha_{<k} = (\alpha_1, \dots, \alpha_{k-1})$. This means that we can gradually build $V_k \in \mathbb{Z}_q(\alpha_{<k})(\alpha_k)$ as

$$V_k = U_{k-1} \cdot \alpha_k^{-1} + V_{k-1} + W_{k-1} \cdot \alpha_k \ ,$$

for all $k = 1, \dots, \tau$, from $k = 1$, with $V_0 = \sum_I x_I y_I = 0$, to $V_\tau = Z(\alpha)$.

Therefore, the special-soundness may use the Schwartz-Zippel lemma since the prover can *first* compute and send commitments[1]

$$c_{u_{k-1}} \in \mathsf{Com}(U_{k-1}) \qquad c_{v_{k-1}} \in \mathsf{Com}(V_{k-1}) \qquad c_{w_{k-1}} \in \mathsf{Com}(W_{k-1})$$

to the verifier *before* receiving back the next scalar $\alpha_k \leftarrow\!\$\, \mathbb{Z}_q$ and iterating with

$$c_{v_k} = c_{u_{k-1}}^{\alpha_k^{-1}} \cdot c_{v_{k-1}} \cdot c_{w_{k-1}}^{\alpha_k} \in \mathsf{Com}(V_k)$$

which can also be computed by the verifier. Starting from $1 = \mathsf{Com}(V_0; 0)$ the process stops with $c_{v_\tau} = \mathsf{Com}(Z) =: E$ after $\tau$ iterations. The prover and the verifier then engage in a simple proof for a product relation between $C, D$ and $E$ ensuring $XY = Z$.

For the extraction of the witness, since the protocol reveals only one opening related to $\{c_i\}_{i=0}^{n-1}$ and one opening related to $\{d_i\}_{i=0}^{n-1}$ we cannot hope to extract

[1] Actually, the verifier can compute the commitment of $V_{k-1}$ itself.

the witness in less than $2^\tau \approx n$ rewinds in the proof of special-soundness. Fortunately, we only need less than $2n$ rewinds to extract the witness.

## 6.3 Proof of Inner Product

**Common Reference String:** ck, where $\mathrm{ck} \leftarrow \mathrm{Gen}(1^\lambda)$. It will be implicit in every use of $\mathsf{Com}$.

**Statement:** $\{(c_i, d_i)\}_{i=0}^{n-1}$, where $c_i, d_i \in \mathcal{C}_{\mathrm{ck}}$, for all $i = 0$ to $n-1$, and $n = 2^\tau$.

**Prover's Witness:** openings $\{(x_i, r_i), (y_i, s_i)\}_{i=0}^{n-1}$ such that $c_i = \mathsf{Com}(x_i, r_i)$ and $d_i = \mathsf{Com}(y_i, s_i)$, for all $i = 0$ to $n-1$, and satisfying $\sum_{i=0}^{n-1} x_i y_i = 0$.

**Initial Round:** common inputs are ck and the statement.

$\boldsymbol{\mathcal{P}} \rightarrow \boldsymbol{\mathcal{V}}$: Pick $\mu_0, \nu_0 \leftarrow\!\$\, \mathbb{Z}_q$ and compute

$$c_{u_0} = \mathsf{Com}(U_0, \mu_0) \ , \qquad U_0 = \sum\nolimits_{i_1^+ \in \{0,1\}^{\tau-1}} x_{0 i_1^+} \cdot y_{1 i_1^+} \ ,$$

$$c_{w_0} = \mathsf{Com}(W_0, \nu_0) \ , \qquad W_0 = \sum\nolimits_{i_1^+ \in \{0,1\}^{\tau-1}} x_{1 i_1^+} \cdot y_{0 i_1^+} \ .$$

Send $c_{u_0}$ and $c_{w_0}$. (We set $V_0 = 0$, $\rho_0 = 0$ so that $c_{v_0} = \mathsf{Com}(V_0, \rho_0) = 1$)

$\boldsymbol{\mathcal{V}} \rightarrow \boldsymbol{\mathcal{P}}$: If $c_{u_0}, c_{w_0} \in \mathcal{C}_{\mathrm{ck}}$, pick and send $\alpha_1 \leftarrow\!\$\, \mathbb{Z}_q$, else abort and output 0.

For later use, already compute $c_{v_1} = c_{u_0}^{\alpha_1^{-1}} \cdot c_{v_0} \cdot c_{w_0}^{\alpha_1}$.

**Iterative Round:** in the $k$-th round the common inputs are ck, the statement as well as the values generated in the previous rounds $\{(c_{u_{k-2}}, c_{w_{i-2}}, \alpha_{i-1})\}_{i=2}^{k}$ and the private prover's inputs are the witness and $\{(\mu_{i-2}, \rho_{i-2}, \nu_{i-2})\}_{i=2}^{k}$.

$\boldsymbol{\mathcal{P}} \rightarrow \boldsymbol{\mathcal{V}}$: Pick $\mu_{k-1}, \nu_{k-1} \leftarrow\!\$\, \mathbb{Z}_q$ and compute

$$c_{u_{k-1}} = \mathsf{Com}(U_{k-1}, \mu_{k-1}) \ , \quad U_{k-1} = \sum\nolimits_{i_k^-, i_k^+, j_k^-} x_{i_k^- 0 i_k^+} \cdot y_{j_k^- 1 i_k^+} \cdot \alpha_{<k}^{i_k^- - j_k^-} \ ,$$

$$c_{w_{k-1}} = \mathsf{Com}(W_{k-1}, \nu_{k-1}) \ , \quad W_{k-1} = \sum\nolimits_{i_k^-, i_k^+, j_k^-} x_{i_k^- 1 i_k^+} \cdot y_{j_k^- 0 i_k^+} \cdot \alpha_{<k}^{i_k^- - j_k^-} \ .$$

Send $c_{u_{k-1}}$ and $c_{w_{k-1}}$. (Compute $\rho_{k-1} = \mu_{k-2}\alpha_{k-1}^{-1} + \rho_{k-2} + \nu_{k-2}\alpha_{k-1}$.)

$\boldsymbol{\mathcal{V}} \rightarrow \boldsymbol{\mathcal{P}}$: If $c_{u_{k-1}}, c_{w_{k-1}} \in \mathcal{C}_{\mathrm{ck}}$, pick and send $\alpha_k \leftarrow\!\$\, \mathbb{Z}_q$, else abort and output 0.

For later use, already compute $c_{v_k} = c_{u_{k-1}}^{\alpha_k^{-1}} c_{v_{k-1}} c_{w_{k-1}}^{\alpha_k}$.

**Penultimate Round:** ($(\tau+1)$-th round) after the first $\tau$ rounds the common inputs are ck, the statement as well as $\{(c_{u_{i-1}}, c_{w_{i-1}}, \alpha_i)\}_{i=1}^{\tau}$ and the private prover's inputs are the witness and $\{(\mu_{i-1}, \rho_{i-1}, \nu_{i-1})\}_{i=1}^{\tau}$. From that point, both $\mathcal{P}$ and $\mathcal{V}$ can deterministically compute

$$C = \prod\nolimits_I c_I^{\alpha^I} = \mathsf{Com}\big(\sum\nolimits_I x_I \alpha^I; \sum\nolimits_I r_I \alpha^I\big) = \mathsf{Com}(X, R)\,,$$

$$D = \prod\nolimits_I d_I^{\alpha^{-I}} = \mathsf{Com}\big(\sum\nolimits_I y_I \alpha^{-I}; \sum\nolimits_I s_I \alpha^{-I}\big) = \mathsf{Com}(Y, S)\,,$$

and $c_{v_\tau} = \mathsf{Com}(V_\tau, \rho_\tau)$ iteratively, as $\mathcal{V}$ already did. It remains to ensure that $XY = V_\tau$ using a standard protocol.

**$\mathcal{P} \to \mathcal{V}$**: First, compute $\rho_\tau = \mu_{\tau-1}\alpha_\tau^{-1} + \rho_{\tau-1} + \nu_{\tau-1}\alpha_\tau$ and $T = \rho_\tau - SX$ such that $E := c_{v_\tau} = D^X \cdot \mathsf{Com}_{\mathrm{ck}}(0, T)$. Then, pick $X', Y', R', S', T' \leftarrow\!\$\, \mathbb{Z}_q$ and compute the commitments

$$C' = \mathsf{Com}(X', R')\,, \quad D' = \mathsf{Com}(Y', S')\,, \quad E' = \mathsf{Com}(YX', SX' + T')\,,$$

so that $E' = D^{X'} \cdot \mathsf{Com}(0, T')$. Send $C', D'$ and $E'$.

**$\mathcal{V} \to \mathcal{P}$**: If $C', D', E' \in \mathcal{C}_{\mathrm{ck}}$, pick and send $\beta \leftarrow\!\$\, \mathbb{Z}_q$, else abort and output 0.

**Final Round:** the common inputs are ck, the statement, $\{(c_{u_{k-1}}, c_{w_{i-1}}, \alpha_i)\}_{i=1}^{\tau}$ as well as $(C', D', E')$ and the private prover's inputs are the witness, the $\tau$ triples $\{(\mu_{i-1}, \rho_{i-1}, \nu_{i-1})\}_{i=1}^{\tau}$, the opening values $(X, Y, R, S, T)$ and $\rho_\tau$ as well as the random scalars $(X', Y', R', S', T')$ and $\beta$.

**$\mathcal{P} \to \mathcal{V}$**: Compute and send the final response as

$$z_x = \beta X + X'\,, \qquad z_y = \beta Y + Y'\,,$$
$$z_r = \beta R + R'\,, \qquad z_s = \beta S + S'\,, \qquad z_t = \beta T + T'\,.$$

Note that $X$ and $Y$ have been computed in the previous round.

**$\mathcal{V} \to \mathcal{P}$**: If $z_x, z_y, z_r, z_s, z_t \in \mathbb{Z}_q$ does not hold, output 0, else perform the last verification: from $\alpha = (\alpha_1, \ldots, \alpha_\tau)$, compute $C = \prod_I c_I^{\alpha^I}$ and $D = \prod_I d_I^{\alpha^{-I}}$, and check whether

$$C^\beta C' = \mathsf{Com}(z_x, z_r)\,, \quad D^\beta D' = \mathsf{Com}(z_y, z_s)\,, \quad E^\beta E' = D^{z_x}\mathsf{Com}(0, z_t)\,,$$

holds or not. If so, output 1, otherwise, output 0.

*Efficiency* The communication complexity of the interactive protocol is $2\tau+3$ commitments and 5 scalars of $\mathbb{Z}_q$ for the prover and $\tau+1$ scalars for the verifier. The size of the transcript $\langle c_{u_0}, c_{w_0}, \alpha_1, \ldots, c_{u_{\tau-1}}, c_{w_{\tau-1}}, \alpha_\tau, C', D', E', \beta, z_x, z_y, z_r, z_s, z_t\rangle$ is $2\tau+3$ commitments and $\tau+6$ scalars. The non interactive version of this $\tau+2$-round protocol based on the Fiat-Shamir heuristic saves the 3 last commitments of the transcript and the $\tau$ scalars of the challenge tuple $(\alpha_1, \ldots, \alpha_\tau)$.

## 6.4 The many-bits case

We turn the proof of inner-product into a proof that $n$ ElGamal ciphertexts $\{(g^{r_i}, h^{v_i + r_i})\}_{i=0}^{n-1}$ encrypt $v_i \in \{0, 1\}$, for all $i+1 \in [n]$. Since the ElGamal encryption is homomorphic, we have $c_i = \mathsf{Com}(v_i; r_i)$ in the previous notation. Also, $d_i := \mathsf{Com}(1; 0)\, c_i^{-1} = \mathsf{Com}(1 - v_i, -r_i)$ is a publicly computable ElGamal encryption of $1 - v_i$, so that $x_i = v_i$ and $y_i = 1 - v_i$ in the previous notation.

Assuming that $n = 2^\tau$, a direct application of the inner product proof only ensures $\sum_{i=0}^{n-1} v_i(1 - v_i) = 0$ while our goal is $v_i(1 - v_i) = 0$, for all $i+1 \in [n]$, and not their sum. Fortunately, by applying the Schwartz-Zippel technique to $y'_i = y_i \cdot \gamma^i$ over $d'_i = d_i^{\gamma^i}$, we see that $\sum_{i=0}^{n-1} v_i(1 - v_i)\gamma^i = 0$ implies, for all $i+1 \in [n]$, $v_i(1 - v_i) = 0$, but with negligible probability $n/q$. In the case that $n$ is not a power of two, we can still pad $x$ and $y'$ with 0's using dummy ElGamal ciphertext $c_i = \mathsf{Com}(0, 0)$ and $d'_i = \mathsf{Com}(0, 0)$. Note that we can ignore these terms when computing the $X$, $Y$, $U_k$ and $W_k$ sums in the proof as they will always result in a 0 value. Therefore, the padding does not increase the cost of the proof. Finally, note that we drop the $D', z_s$ and $z_s$ entries in the proof for the many-bits case. They are used to prove the knowledge of $y$, but here $y$ is directly derived from $x$.

*Efficiency.* Computing the log-based proof requires $4 \log n + 4$ exponentiations in $\mathbb{Z}_p^*$: 4 in each round to compute $c_u$ and $c_w$ and 4 to compute $C'$ and $E'$. However, we now need to compute $n(\tau + 8) + 2\tau + 7$ multiplications and $n(\tau + 5) + 4\tau + 8$ additions in $\mathbb{Z}_q$, which now dominate the cost asymptotically since $\tau = \log n$. Finally, the size of the proof is $4\tau$ elements in $\mathbb{Z}_p^*$ and 4 elements in $\mathbb{Z}_q$.

## 7 Benchmarking

We implemented the algorithms described above in Python, and executed them on an AMD 3990X processor with the turbo boost technology disabled, using the Python 3.8.10 interpreter, and gmpy2 2.1.0. All the running times listed below are in milliseconds. The implementation of the schemes and the benchmarks can be found on this repository: `https://github.com/uclcrypto/many01proofs`.

### 7.1 Precomputation and exponentiations

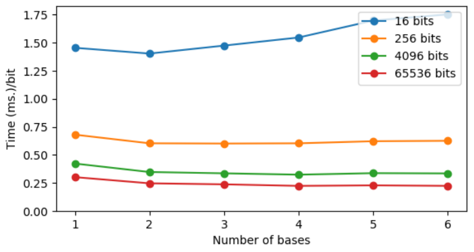
We first tested the time needed to precompute and compute 1000 exponentiations in the ElectionGuard default group ($|p| = 4096$, $|q| = 256$).

As a baseline, computing 1000 modular exponentiations using gmpy2.powmod takes 1560 ms. Figure 1.(a) shows the precomputation and computation time for various values of the precomputation parameter $k$. The speed-up factors of the exponentiation, compared to gmpy2.powmod are quite important: they range from a factor 4 when $k = 4$ to a factor 12 when $k = 13$.

Depending on the application context, the precomputation time may need to be taken into account. This question is particularly interesting in the case of multi-ElGamal encryption where, for a given number of bits to verifiably encrypt, we may wonder which $(k, m)$ pair leads to an optimal running time, as discussed in Section 4.

This is explored in Figure 1.(b), in which we show, for a number $n$ of bits to be encrypted with $n \in \{2^4, 2^8, 2^{12}, 2^{16}\}$, the computation time/bit for $m \in [1, 6]$, selecting the optimal $k$ every time. The lines are surprisingly flat: what is gained by multi-ElGamal is lost in the fixed-base exponentiations. The maximum benefits of multi-ElGamal are around 25%, much lower than the factor of almost 2 that was hoped for, and this gain is reached around $m = 4$ for our 3 highest values of $n$. So, picking $m = 4$ independently of $n$ might be a reasonable choice, should multi-ElGamal encryption be adopted. The picture is of course different is the precomputation time does not need to be taken into account: there, multi-ElGamal keeps its expected efficiency benefits.

| $k$ | 0 | 4 | 8 | 13 |
|---|---|---|---|---|
| precomp. | - | 6.1 | 51 | 1042 |
| 1000 exp | 1560 | 387.6 | 204.2 | 131 |



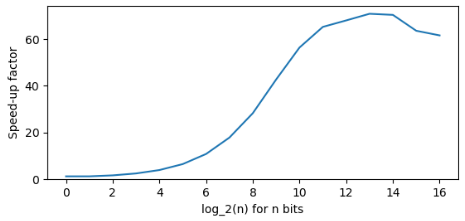
(a) (b)

Fig. 1: (a) Precomputation time and time to compute 1000 exponentiations. (b): evolution of the time needed to encrypt votes with optimal choice of $k$ and various choices of the number of multi-ElGamal bases.

### 7.2 Verifiable bit encryption using linear techniques

As a second step, we explore in Figure 2(a) the efficiency of our proof techniques with linear complexity. For the same values of $n$ as above, a choice of $m = 4$ for the multi-ElGamal encryption, and optimal values of $k$, we explore the time needed for performing the precomputation, encryption, product proof (from Table 2), and batch proof (from Table 3). We can make a few observations from this figure: (i) As expected, the optimal $k$ grows with $n$, and so does the time spent in precomputation. But the proportion of the time spent in precomputation decreases when $n$ increases, which illustrates the decreasing returns of increasing $k$. (ii) While $n$ is multiplied by a factor 16 from line to line, the computation time is only multiplied by a factor around 10, thanks to the amortization coming from an increased amount of precomputation. (iii) The cost of our batch proof is essentially equal to the cost of the multi-ElGamal encryption.

| $n$ | $k$ | Pre. | Multi-ElGamal | Prod. Proof | Batch Proof |
|---|---|---|---|---|---|
| 16 | 2 | 12 | 13 | 33 | 16 |
| 256 | 5 | 51 | 103 | 270 | 107 |
| 4096 | 7 | 150 | 1214 | 3187 | 1256 |
| 65536 | 11 | 1573 | 13022 | 34353 | 13605 |



(a) (b)

Fig. 2: (a) Running time with linear techniques. (b) Speed-up given by the log proof for $n \in \{2^i | i \in [16]\}$.

### 7.3 Verifiable bit encryption using logarithmic proof techniques

Eventually, Figure 2(b) shows the speed-ups obtained in an implementation of our logarithmic proof of Section 6, compared to the baseline Chaum-Pedersen disjunctive proof (the batched proof is faster by a factor $\approx 2.5$ when $n$ is big enough). Both proofs take advantage of the fixed base precomputation, which is needed for the ElGamal encryption.

Here, and despite a higher asymptotic complexity due to the $n \log(n)$ multiplications needed in $\mathbb{Z}_q$, the log proof provides dramatic speed improvements, by a factor up to 70 when around 8000 proofs need to be computed, making the cost of the proof computation negligible compared to the one of encryption. The gains are already there for a small number of proofs: we obtain a speedup by a factor 2.5 for 8 proofs and a factor 18 for 128 proofs, about the size of a ballot. If one needs to compute a large number of 0-1 proofs, it may be more convenient and efficient to compute many logarithmic proofs by batches of a size between $2^{10}$ and $2^{14}$ for instance.

## 8 Conclusions

We proposed various techniques that could be used to increase the speed of verifiably encrypting bits, both at the arithmetic level (modular exponentiations) and at the protocol level (encryption and ZK proofs), compared to the usual implementation of the protocol of Cramer et al. [10].

Fixed base exponentiation techniques showed dramatic speed improvements. But, interestingly, these techniques reduced the potential benefits associated to the use of multi-ElGamal encryption, which requires fewer exponentiations but more bases: when the precomputation time is accounted for, we observe essentially no benefit in using more than $m = 4$ ElGamal public keys.

We then turned to the ZK proofs. First, we observed that changing a base in the ElGamal encryption, which is of no consequence there, brings a 25% speed-up on the traditional proof of Cramer et al. Switching to product proofs rather than disjunctive proofs, we observed no benefit for a single ElGamal ciphertext, but new possibilities for batching that brought a speed-up by a factor close to 3 on the proof computation for multi-ElGamal ciphertexts: the total cost of the encryption and proof comes close to 2 exponentiations per encrypted bit.

Eventually, we turned to a different proof strategy, in the line of the many recent protocols aiming at bringing short (logarithmic size) proofs. Here, and despite a worse asymptotic complexity, we again observed very important speed-ups, making the cost of the proof almost negligible compared to the cost of an ElGamal encryption, leading again to a cost close to 2 exp./encrypted bit.

This leaves a natural question for further works: can we go below a complexity of around 2 exp./bit for DL based protocols and parameters useful for an election? Another question comes from the choice of the group, $\mathbb{Z}_p^*$, which is by far the most common choice in current voting system implementations: how would our benchmarks evolve if ECC were considered?

**Acknowledgements.** This research was supported in part by the FNRS project SeVote and by a Microsoft Research Award. Henri Devillez is supported by a FRIA grant and Thomas Peters is a research associate of the FNRS. We thank Josh Benaloh, Michael Naehrig and Dan Wallach for numerous interesting discussions and useful feedback about this work.

## References

1. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.: Electing a university president using open-audit voting: Analysis of real-world use of helios. In: 2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09. USENIX Association (2009)
2. Bellare, M., Goldwasser, S.: Verifiable partial key escrow. In: CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security. pp. 78–91. ACM (1997)
3. Benaloh, J., Byrne, M.D., Eakin, B., Kortum, P.T., McBurnett, N., Pereira, O., Stark, P.B., Wallach, D.S., Fisher, G., Montoya, J., Parker, M., Winn, M.: Star-vote: A secure, transparent, auditable, and reliable voting system. In: 2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13. USENIX Association (2013)
4. Benaloh, J., Foote, K., Stark, P.B., Teague, V., Wallach, D.S.: Vault-style risk-limiting audits and the inyo county pilot. IEEE Secur. Priv. **19**(4), 8–18 (2021)
5. Benaloh, J.C., Yung, M.: Distributing the power of a government to enhance the privacy of voters (extended abstract). In: Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing. pp. 52–62. ACM (1986)
6. Bernstein, D.J.: Pippenger's exponentiation algorithm. `https://cr.yp.to/papers/pippenger.pdf` (Jan 2002)
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Advances in Cryptology - CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer (1992)
8. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A simple private and verifiable electronic voting system. In: Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows. LNCS, vol. 11565, pp. 214–238. Springer (2019)
9. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Advances in Cryptology - CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer (1994)
10. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Advances in Cryptology - EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer (1997)
11. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) Public Key Cryptography, PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer (2001)
12. Damgård, I., Jurik, M.: A length-flexible threshold cryptosystem with applications. In: Information Security and Privacy, ACISP 2003. LNCS, vol. 2727, pp. 350–364. Springer (2003)
13. ElectionGuard: `https://www.electionguard.vote/` (May 2022)
14. Frederiksen, T.K., Lindell, Y., Osheter, V., Pinkas, B.: Fast distributed RSA key generation for semi-honest and malicious adversaries. In: Advances in Cryptology - CRYPTO 2018. LNCS, vol. 10992, pp. 331–361. Springer (2018)

15. gmpy: gmpy2 module. `https://github.com/aleaxit/gmpy`
16. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Advances in Cryptology - CRYPTO 2009. LNCS, vol. 5677, pp. 192–208. Springer (2009)
17. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Advances in Cryptology - EUROCRYPT 2015. LNCS, vol. 9057, pp. 253–280. Springer (2015)
18. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag, Berlin, Heidelberg (2003)
19. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract). In: Advances in Cryptology - EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer (1991)
20. Pippenger, N.: On the evaluation of powers and related problems (preliminary version). In: 17th Annual Symposium on Foundations of Computer Science, 1976. pp. 258–263. IEEE Computer Society (1976)
21. Ramchen, K., Culnane, C., Pereira, O., Teague, V.: Universally verifiable MPC and IRV ballot counting. In: Financial Cryptography and Data Security - FC 2019. LNCS, vol. 11598, pp. 301–319. Springer (2019)
22. Sandler, D., Derr, K., Wallach, D.S.: Votebox: A tamper-evident, verifiable electronic voting system. In: van Oorschot, P.C. (ed.) Proceedings of the 17th USENIX Security Symposium. pp. 349–364. USENIX Association (2008)
23. Schoenmakers, B.: Some efficient zeroknowledge proof techniques. In: Workshop on cryptographic protocols (2001)
24. Wallach, D.: Anyscale connect: Encrypting and tabulating big elections. `https://www.youtube.com/watch?v=m7r33EuN6Zw` (Dec 2020)
25. Wikström, D.: Verificatum. `https://www.verificatum.org/` (May 2022)
26. Wu, T.: Rsa and ecc in javascript. `http://www-cs-students.stanford.edu/~tjw/jsbn/` (2009)