How to fake zero-knowledge proofs, again

Véronique Cortier, Pierrick Gaudry, Quentin Yang

To cite this version:

Véronique Cortier, Pierrick Gaudry, Quentin Yang. How to fake zero-knowledge proofs, again. E-Vote-Id 2020 - The International Conference for Electronic Voting, Oct 2020, Bregenz / virtual, Austria. hal-02928953

HAL Id: hal-02928953

https://inria.hal.science/hal-02928953

Submitted on 3 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

How to fake zero-knowledge proofs, again

Véronique Cortier[1], Pierrick Gaudry[1], and Quentin Yang[1,2]

[1] Université de Lorraine, CNRS, Inria, Nancy, France
[2] École Polytechnique, Palaiseau, France

Abstract. In 2012, Bernhard et al. showed that the Fiat-Shamir heuristic must be used with great care in zero-knowledge proofs. We explain how, in the Belenios voting system, while not using the weak version of Fiat-Shamir, there is still a gap that allows to fake a zero-knowledge proof in certain circumstances. Therefore an attacker who corrupts the voting server and the decryption trustees could break verifiability.

1 Presentation of the result

Context. Zero-knowledge proofs are heavily used in e-voting protocols. A voter typically proves that her vote belongs to a set of valid choices and authorities show that they correctly decrypted the result. A standard way to move from an interactive proof to a non-interactive one is the Fiat-Shamir heuristic [5] where the randomness of the verifier is simulated by hashing the inputs of the prover. Bernhard et al. [2] have highlighted that great care must be taken in the part of the inputs that are hashed. In particular, for proofs on messages encrypted using ElGamal, they showed that it is not sufficient to hash the commitment (weak Fiat-Shamir). Instead, the full context must be hashed (strong Fiat-Shamir), otherwise decrypting authorities could, for example, change the election result yet producing a valid proof of correct decryption, in the context of the Helios [1] protocol. A similar idea [7] has been used in 2019 to attack the voting protocol of Scytl that were to be deployed in Switzerland.

Our contribution. Belenios [4] is a variant of Helios that prevents ballot stuffing and that has been used in more than 500 elections since 2018. In Belenios, a version of the Fiat-Shamir heuristic is used that does not fully follow the strong Fiat-Shamir as described in [2], since, apart from the commitment, only the ciphertext is hashed, and not the other parts of the context. In particular, like in Helios, the generator g of the group is, in principle, a parameter that can be chosen to be different for each election. This gives one degree of freedom for the authority in charge of setting up the election. Another degree of freedom can come from the public key when the decryption authorities collude with the authority. Combining both, we show that an attacker can produce a ciphertext and a valid proof of set membership, while the ciphertext will actually decrypt to an arbitrary vote controlled by the attacker. Compared to [2], the attack is slightly more involved since in Belenios, the whole ciphertext is hashed in the zero-knowledge proof, instead of just the commitment in Helios. In particular, our attack requires to adversarially select the group generator.

To illustrate this weakness, we consider the context of an election organized with Belenios, where voters select at most one candidate over a list of k choices. An attacker who can choose the group generator and the decryption key will be able to produce a full ballot that would be considered as valid but contains an arbitrary value. Interestingly, the "proof" that the ballot is valid involves several zero-knowledge proofs, but even if the attacker can fake only a single proof of set membership, it can be used several times to construct an entirely valid fake ballot.

How does it work? The first step is to create an ElGamal ciphertext with a fake proof of set membership, following the specification of Belenios for this zero-knowledge proof [6]. Without going into the details (see next section), we start from a pair of elements (x, y) for which the discrete logarithms are known w.r.t. a generator γ, and then re-interpret them as an ElGamal ciphertext using another generator $g = \gamma^a$. The value a can be chosen by the attacker and gives a degree of freedom that allows to construct all the data needed for a fake proof that the encrypted value is either 0 or 1. In this construction, the secret key that serves to decrypt the ballots must be known from the attacker. If it can be chosen by the attacker, this gives an additional degree of freedom to force a certain decryption value instead of having a random one when decrypting (x, y).

The second step is to use this ciphertext (x, y) to build a fake ballot. The structure of a ballot is a list of k encrypted bits, each of them telling whether or not the corresponding candidate is chosen. There are individual proofs that these are indeed encrypted bits, and an overall proof that the homomorphic sum of these ballots is also in $\{0, 1\}$. The construction uses (x, y) as a fake bit-encryption for the first candidate, and produces genuine encrypted bits for the others, but takes advantage of the randomness to enforce the homomorphic sum of all of them to be (x, y) as well. Therefore the fake proof of set membership for (x, y) can be used twice and the ballot looks valid.

Impact. In a setting where the attacker controls the server and the decryption trustees, our attack breaks verifiability since the result of the election can be arbitrarily modified by the attacker. This attack has been missed in the security proof of [3] because the group generator was assumed to be a fixed, known, parameter.

The fix is pretty simple: the strong Fiat-Shamir heuristic should be used, that is the group generator and the public key of the election should be included in the hash, as advised in [2] for Helios. Interestingly, the current implementation of Helios still uses the weak Fiat-Shamir heuristic and is hence still subject to the attacks mentioned in [2] as well as our attack where the attacker plays with the group generator instead of the ciphertext.

2 Technical details

For the sake of completeness, we provide here the full details on how to build a fake ballot with a valid proof by choosing the group generator appropriately, in the context of a Belenios election where voters choose at most one candidate among k choices.

We start by recalling how to build a zero-knowledge proof of set membership, using the Fiat-Shamir heuristic. This relies on a hash function that maps to $\mathbb{Z}_q$; we denote it by hash. We consider a group G of prime order q and we write $w \in_r \mathbb{Z}_q$ to say that w is drawn uniformly at random in $\mathbb{Z}_q$, independently of all the other random choices. We follow the Belenios specification which uses an intermediate between weak and strong Fiat-Shamir.

Set membership proof. Let g be a group generator in G, and let $h = g^s$ be a public key corresponding to a secret key $s \in \mathbb{Z}_q$. Given an ElGamal encryption $(x, y) = (g^\alpha, g^m h^\alpha)$ of a cleartext m, and given α, the prover wishes to prove that m belongs to some fixed set $\{m_1, \cdots, m_n\}$.

The prover proceeds as follows:

1. Let i in $[1, n]$ be such that $m = m_i$. Let $w \in_r \mathbb{Z}_q$ and compute $A_j = g^w$ and $B_j = h^w$.
2. For any $j \neq i$, $1 \leq j \leq n$, let $\sigma_j, \rho_j \in_r \mathbb{Z}_q$ and compute $A_j = g^{\rho_j} x^{-\sigma_j}$ and $B_j = h^{\rho_j} (y/g^{m_j})^{-\sigma_j}$.
3. Let $c = \mathsf{hash}(x||y||A_1||B_1||\cdots||A_n||B_n)$ and compute $\sigma_i = c - \sum_{j \neq i} \sigma_j$ and $\rho_i = w + \alpha\sigma_i$.

The set membership proof is then

$$\pi = (\sigma_1, \rho_1), \ldots, (\sigma_n, \rho_n).$$

To check the validity of this proof, the verifier proceeds as follows:

1. Compute $A_i = g^{\rho_i} x^{-\sigma_i}$ and $B_i = h^{\rho_i} (y/g^{m_i})^{-\sigma_i}$ for all $1 \leq i \leq n$.
2. Check that $\mathsf{hash}(x||y||A_1||B_1||\cdots||A_n||B_n) = \sum_{i \in [1,n]} \sigma_i$.

The security properties of this Σ protocol, *i.e.* completeness, zero-knowledge and special-soundness are very classical and can be found for instance in [6].

Forging a set membership proof in {0, 1}. We show here how an attacker can select a group generator g and a secret key s such that she can forge a ciphertext (x, y) and a proof π that passes the validity check while (x, y) is an encryption of a message V chosen by the attacker.

The construction proceeds as follows:

1. Let γ be a generator of G.
2. Let $\alpha, \beta, r_1, r_2, r_3, r_4 \in_r \mathbb{Z}_q$.
3. Let $x = \gamma^\alpha$, $y = \gamma^\beta$.
4. Compute $c = \mathsf{hash}(x||y||\gamma^{r_1}||\gamma^{r_2}||\gamma^{r_3}||\gamma^{r_4})$.

Recall that one has to find $g, h, \sigma_1, \rho_1, \sigma_2, \rho_2$ such that $\sigma_1 + \sigma_2 = \mathsf{hash}(x||y|| g^{\rho_1} x^{-\sigma_1}||h^{\rho_1} y^{-\sigma_1}||g^{\rho_2} x^{-\sigma_2}||h^{\rho_2} (y/g)^{-\sigma_2})$. In order to do so, one can choose $a, s \in \mathbb{Z}_q$ such that $g = \gamma^a$ and $h = g^s$, along with the corresponding $\sigma_1, \rho_1, \sigma_2, \rho_2$ such that $\mathsf{hash}(x||y||g^{\rho_1}x^{-\sigma_1}||h^{\rho_1}y^{-\sigma_1}||g^{\rho_2}x^{-\sigma_2}||h^{\rho_2}\left(y/g\right)^{-\sigma_2}) = c$. This leads to the following equations

$$\begin{cases} g^{\rho_1}x^{-\sigma_1} = \gamma^{r_1}, \\ h^{\rho_1}y^{-\sigma_1} = y^{r_2}, \\ g^{\rho_2}x^{-\sigma_2} = \gamma^{r_3}, \\ h^{\rho_2}\left(y/g\right)^{-\sigma_2} = \gamma^{r_4}, \\ \sigma_1 + \sigma_2 = c. \end{cases}$$

Using the logarithm in base γ, we obtain the following system of equations to be verified modulo q:

$$\begin{cases} a\rho_1 - \alpha\sigma_1 = r_1 \\ as\rho_1 - \beta\sigma_1 = r_2 \\ a\rho_2 - \alpha\sigma_2 = r_3 \\ as\rho_2 - \sigma_2(\beta - a) = r_4 \\ \sigma_1 + \sigma_2 = c \end{cases} \text{, hence } \begin{cases} \rho_1 = \frac{r_1+\alpha\sigma_1}{a} \\ \sigma_1 = \frac{r_2-sr_1}{\alpha s-\beta} \\ \rho_2 = \frac{r_3+\alpha\sigma_2}{a} \\ \sigma_2 = \frac{r_4-sr_3}{\alpha s-\beta+a} \\ \frac{r_2-sr_1}{\alpha s-\beta} + \frac{r_4-sr_3}{\alpha s-\beta+a} = c. \end{cases}$$

The first four equations give explicit formulas which allows one to derive $\sigma_1, \rho_1, \sigma_2$ and ρ_2 from a and s while the last one gives a sufficient condition for the proof to be accepted. In addition, (x, y) decrypts into g^V if and only if $a = \frac{\beta-\alpha s}{V}$. Therefore, one can choose s as the solution of the following equation that becomes linear in s after clearing denominators:

$$\frac{r_2 - sr_1}{\alpha s - \beta} + \frac{r_4 - sr_3}{(\alpha s - \beta)(1 - \frac{1}{V})} = c.$$

Consequently, the remaining of the procedure consists of the following steps:

5. Let $s = \left(\beta c + r_2 + \frac{Vr_4}{V-1}\right)\left(\alpha c + r_1 + \frac{Vr_3}{V-1}\right)^{-1}$ and $a = \frac{\beta-\alpha s}{V}$.
6. Let $\sigma_1 = \frac{r_2-sr_1}{\alpha s-\beta}$, $\rho_1 = \frac{r_1+\alpha\sigma_1}{a}$, $\sigma_2 = \frac{r_4-sr_3}{\alpha s-\beta+a}$ and $\rho_2 = \frac{r_3+\alpha\sigma_2}{a}$.
7. Return the elements $g = \gamma^a$ as a generator, $h = g^s$ as a public key, s a secret key, (x, y) as a ciphertext, and $\pi = (\sigma_1, \rho_1), (\sigma_2, \rho_2)$ as a fake set membership proof for (x, y).

The computations of s, σ_1, ρ_1, σ_2, ρ_2, and a involve divisions by quantities that could in principle be zero. However, these are random elements in $\mathbb{Z}_q$, so that the probability of these events to occur is negligible (and one could still start again with other randoms).

As explained above, the verifier will accept the proof π, yet (x, y) does not encrypt 0 nor 1 but V, where V is chosen by the attacker. Thus, she can use this approach to build a ciphertext that encrypts a particular value of her choice or that could be used to add or subtract some amount of votes.

Forging a fake Belenios ballot. A full Belenios ballot for an election with k candidates is actually of the form $(x_1, y_1), \ldots, (x_k, y_k), \pi_0, \pi_1, \ldots, \pi_k$ where:

– π_0 is a proof that $\prod_{i=1}^{k}(x_i, y_i)$ is the encryption of 0 or 1 (the voter should select at most one candidate);

– for $i \geq 1$, π_i is a proof that (x_i, y_i) is the encryption of 0 or 1 (the voter can choose the ith candidate, or not).

In Belenios, a ballot is also signed by a credential but this part is irrelevant here (the adaptation is straightforward).

To forge a ballot that looks valid but contains an arbitrary value V, we proceed as follows:

– let $(x_1, y_1) = (x, y)$ and $\pi_1 = \pi$ where (x, y) is the forged ciphertext for which we have a valid proof π that (x, y) is the encryption of 0 or 1, while it is the encryption of V;
– pick random $\alpha_2, \ldots, \alpha_{k-1} \in_r \mathbb{Z}_q$ and let $(x_i, y_i) = (g^{\alpha_i}, h^{\alpha_i})$ be an encryption of 0, and compute π_i a (honest) proof that (x_i, y_i) encrypts 0 or 1;
– let $\alpha_k = -\sum_{i=2}^{k-1} \alpha_i$ and $x_k = g^{\alpha_k}$ et $y_k = h^{\alpha_k}$, so that (x_k, y_k) is an encryption of 0, and compute π_k a (honest) proof that (x_k, y_k) encrypts 0 or 1;
– Finally, $\prod_{i=1}^{k}(x_i, y_i) = (x, y)$ hence we can simply take $\pi_0 = \pi$ for the last proof.

The forged ballot will pass all the tests hence will be accepted. However, (x_1, y_1) is an encryption of V. Since only the homomorphic sums of all the votes for each candidate are decrypted, the tally will show V additional votes for the first candidate, where V is arbitrarily chosen by the attacker.

The attacker could also choose to set V to a huge value, so that decryption, which is based on a small discrete-logarithm computation will run forever.

References

1. Adida, B.: Helios: Web-based open-audit voting. USENIX Security 2008. pp. 335–348. USENIX Association, 2008
2. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, 2012.
3. Cortier, V., Dragan, C.C., Dupressoir, F., Schmidt, B., Strub, P.Y., Warinschi, B.: Machine-checked proofs of privacy for electronic voting protocols. 2017 IEEE Symposium on Security and Privacy. pp. 993–1008. IEEE, 2017.
4. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A simple private and verifiable electronic voting system. Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows. LNCS, vol. 11565, pp. 214–238. Springer, 2019
5. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, 1987.
6. Gaudry, P.: Some ZK security proofs for Belenios. `https://hal.inria.fr/hal-01576379`, 2017, informal note
7. Lewis, S.J., Pereira, O., Teague, V.: Trapdoor commitments in the SwissPost e-voting shuffle proof. `https://people.eng.unimelb.edu.au/vjteague/SwissVote`, 2019, blog entry