A Note on the Security of GG18

Nikolaos Makriyannis[*] Udi Peled[*]

Abstract

We present attacks for information disclosure in two variants of the Gennaro and Goldfeder (CCS'2018) protocol for threshold-ECDSA signing, including the "full" variant prescribed by the paper. Although we could not expand this leakage into full blown key-extraction for neither variant, we consider this leakage to be highly problematic. The two attacks target the so-called multiplicative-to-additive phase (MtA) allowing the parties to perform two-party multiplication. Lastly, we propose simple remediation steps to foil the attacks.

Contents

Contents

1 Introduction

Threshold-ECDSA is one of the most popular application areas of practical-MPC, owing its popularity to use of ECDSA in many cryptocurrencies. One of the earliest and more popular protocols for achieving practical full-threshold maliciously secure ECDSA is the protocol of Gennaro and Goldfeder [3] (hereafter refered to as GG18). In this document, we identify vulnerabilities in two different variants of the GG18 protocol in which the adversary succeeds in obtaining leakage of the private key by means of an attack.

The first and more easily fixable vulnerability is on the protocol's variant where the authors suggest omitting some expensive zero-knowledge proofs. The authors conjecture that the protocol remains secure even when the proofs are omitted.[1] We show that in this variant of the protocol, where the proofs are omitted, an attacker may probe an arbitrary number of bits of the honest-party's secret key-share. In fact, *in certain*

[*]Fireblocks. `nikos@fireblocks.com`, `udi@fireblocks.com`

[1]While the authors were aware that omitting the proof may result in some leakage, this conjecture presumed that (and may be reasonable when) the protocol terminates as soon as even a single failure occurs. This assumption is not unrealistic in a real-world setting.

cases, depending on the implementation, we show how the attacker may learn the entirety of the secret key by sequentially probing *all* of the secret's bits.

Our second attack is on the "full" version of the GG18 protocol that includes the range proofs (and also for GG20, the follow-up protocol). We present an adversary which is able to extract the most significant bits of the signature nonce's inverse (also known as the ephemeral key-inverse). This leakage is not accounted for in the security analysis and it is a clear bug for the security proof. Though we couldn't harness this leakage to a full blown private key extraction attack, guessing the key using this leakage seems related to the *inverse hidden number problem* [1, 5], a known variant of the well studied hidden numbers problem. Thus, advances in the inverse hidden-number problem may lead to potentially devastating (future) attacks.[2]

1.1 Overview of GG18 and the Attacks

For simplicity of exposition, we focus on the 2-out-of-2 threshold case, where the adversary controls one of the two parties. The more general case of t-out-of-n threshold is a straightforward generalization (though there is some loss in the attack's success probability). We assume some familiarity with the threshold ECDSA literature and the notation from GG18.

Let $\mathcal{P}_1, \mathcal{P}_2$ denote the parties in an execution of the protocol, and recall that after the multi-party key-generation phase, the private key $x \in \mathbb{Z}_q$ is defined as $x = x_1 + x_2 \mod q$, where q is the prime order of the underlying elliptic curve (e.g. of secp256k1), and x_1 and $x_2 \in \mathbb{Z}_q$ are the secret shares of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. Let N denote the Paillier public key of $\mathcal{P}_1$ (known to both parties), which is chosen many orders of magnitude greater than q (e.g. $|N| = 8 \cdot |q|$).

The cornerstone of the signing protocol is the so-called MtA (Multiplication-to-Addition) sub-protocol, where $\mathcal{P}_i$ sends $\mathcal{P}_j$ the Paillier ciphertext $\mathsf{enc}_N(k_i)$, which encrypts her share of the (inverse-nonce) k.[3] Then, party $\mathcal{P}_j$ responds with $\mathsf{enc}_N(k_i \cdot x_j + \beta)$ (for random β) using the homomorphic properties of Paillier encryption. In the end, $\mathcal{P}_i$ decrypts the received ciphertext and calculates $\alpha = k_i \cdot x_j + \beta \mod q$, which is used later on in the protocol.

The crux of both attacks is the discrepancy between the reduction modulo q after decrypting (for obtaining α), and the reduction modulo N that happens inherently for the plaintext inside the encryption. In an honest execution, the reduction modulo N does not occur, because the parties are instructed to choose *small inputs*. However, if the parties choose *large* values, then it is likely that the reduction modulo N interferes with the reduction modulo q which leads to a faulty signature and/or a failed execution of the protocol. Interestingly, an attacker may correlate this failure with the inputs of the honest party (thus the protocol succeeds/fails only if the honest party's secret input satisfies certain condition). In turn, this allows the adversary to effectively learn (parts of) the honest party's secret.

To address the above, the standard technique (also in the "full" protocol of GG18) involves both parties proving in zero-knowledge that their inputs (k_i and x_j) are chosen from a small domain, via *range proofs*. However, as mentioned earlier, the authors conjectured that removing these proofs does not (substantially) affect the security of the protocol, because the leakage is deemed tolerable.

In this document, we show that both variants of the protocol are prone to non-trivial leakage.

Key Leakage from Variant w/o the Range Proofs. We now show how a malicious $\mathcal{P}_1$ can extract the ℓ least significant bits (LSBs) of the $\mathcal{P}_2$'s secret x_2 (write $b_{\ell-1} \dots b_0$ for the ℓ LSB) as follows: $\mathcal{P}_1$ uses input $f = 2^{-\ell} \mod N$ as input to the MtA (sending $\mathsf{enc}_N(f)$ instead of $\mathsf{enc}_N(k_1)$), and guesses that $b_{\ell-1} \dots b_0 = z \in [0, 2^\ell - 1]$. After decrypting the MtA "response" $\alpha = \mathsf{enc}_N(f \cdot x_2 + \beta)$ from $\mathcal{P}_2$, the attacker resets $\alpha := [\alpha - f \cdot z \mod N] + 2^{-\ell} \cdot z \mod q$ and carries on the protocol using input $k_1 = 2^{-\ell} \mod q$. It is not hard to see that the protocol will succeed (and the signature will verify) if and only if $b_{\ell-1} \dots b_0 = z$, and thus the attacker learns the least significant bit if the protocol terminates correctly.

After extracting the ℓ LSBs of x_2, the next ℓ bits $b_{2\ell-1} \dots b_\ell$ can be extracted by a similarly contrived f' ($= 2^{-2\ell} \mod N$), using the knowledge of the already known $b_{\ell-1} \dots b_0$ to adjust its guess of the 2ℓ LSBs. This process can be iterated until the attacker extracts all the bits of x_2.

[2]For leakage of the (standard, non-inverse) nonce, we mention that the key can be retrieved via the hidden-numbers problem with as few as 40 signatures and 8-bit leakage each (for curves of size $\approx 2^{256}$, e.g. Bitcoin's curve. For smaller curves, the key can be recovered with even smaller leakage/fewer signatures).

[3]The nonce is sometimes referred to as the ephemeral key.

We remark that this attack succeeds only if the execution of the MtA for k_i, x_j is independent of the second MtA (for k_i, γ_j, where γ's are used to mask the ephemeral key) in the protocol. Otherwise, if the same $\mathsf{enc}_N(k_i)$ is used in both homomorphic evaluations by $\mathcal{P}_2$, the failure of the protocol is not correlated with the bits of x (because of the entropy introduced by the γ's) and the attack is foiled. For efficiency reasons, a reasonable implementation of the protocol will send the same $\mathsf{enc}_N(k_i)$ for both MtA's. This restriction, however, does not appear anywhere in the security analysis, so, in principal, we consider the aforementioned process as a legitimate attack on the protocol.

Inverse-Nonce Leakage for the Full Protocol. Our second attack extracts the ℓ most significant bits (MSB) of the (inverse) nonce in the "full" protocol *with the range proofs*. We now show how a malicious $\mathcal{P}_2$ can extract the ℓ most significant bits (MSBs) of the $\mathcal{P}_1$'s (inverse) nonce k_1 (write $b_{255} \ldots b_{255-\ell+1}$ for the ℓ MSB) as follows: In the MtA for k_1, γ_2, an honest $\mathcal{P}_1$ first sends $\mathsf{enc}_N(k_1)$ (with the valid inverse nonce share k_1) $\mathcal{P}_2$ replies with $\mathsf{enc}_N(k_1 \cdot \gamma_2 + \beta)$ for maliciously chosen $\gamma_2 = 1$ and $\beta = N - 2^{256-\ell}(z+1) \mod N$ (guessesing that $z \in \left[0, 2^\ell - 1\right]$ as the ℓ MSB of k_1). Notice that the small γ_2 will pass the ZK range proof verification done by $\mathcal{P}_1$. The attacker carries on the protocol using the chosen inputs. It is not hard to see that the protocol will succeed if and only if the ℓ MSB of k_1 are exactly z, and thus (together with knowing k_2) the attacker learns the MSB of the (inverse) ephemeral key $k = k_1 + k_2$ if the protocol terminates correctly.

Because a fresh nonce is chosen with every signature generation, the above process cannot be iterated to guess the entire ephemeral key. However, as mentioned earlier, by collecting enough leakage/signatures, we speculate that it may be possible to mount a lattice attack via the *inverse hidden number problem* (though such an attack appears impractical, given the current state of the art).

Nevertheless, the above leakage does not appear anywhere in the security analysis of GG18, and we consider it highly problematic given how sensitive discrete log based signatures are to leakage of the (standard, non-inverse) ephemeral key.

Remediation Techniques. We propose the following remediation plan. First, it goes without saying that the range proofs of GG18 cannot be omitted. Second, for the attack on the full version of GG18 , we suggest that the MtA sender (the one providing the k_i) to reject (abort) the protocol if $\alpha \notin [\varepsilon, N - \varepsilon]$ for a randomly chosen $\varepsilon \in [0, \sqrt{N}]$ (it is stressed that a *new* ε is chosen for each MtA). Effectively, introducing the noisy ε decorrelates the aborting-event with the input of the honest party.

An alternative more robust remediation strategy is adding zero-knowledge range proofs wherever needed (for example also for the β in the MtA response), as is done in the CMP protocol [2].

1.2 Organization of the Note

In the remainder, we present the technical material of this note. Since our contributions (attacks) may have several application areas (not only threshold ECDSA), we opt to cast our attacks as "potential vulnerabilities for Paillier-based two-party multiplication". In the next section (Section 2), we give a high-level technical overview. In Section 3, we introduce notation and technical background. In Section 4, we present our attacks.

2 Attacking Paillier-Based 2PC Multiplication

Two-party multiplication allows a pair of distrusting parties to calculate additive shares of a product $x \cdot y$, where one party holds secret input x and the other party holds secret input y. Multiplication is a fundamental building block of arithmetic MPC (analogous to OT in Boolean MPC) and all non-trivial tasks can be reduced to 2PC multiplication. In this note, we visit the folklore technique of using *additive homomorphic encryption* (henceforth AHE) to instantiate multiplication; Alice (the initiator) encrypts her input under her own key, and sends the resulting ciphertext to Bob (the responder). Then, Bob samples random noise σ and calculates an encryption of $x \cdot y + \sigma$ using the homomorphic properties of the encryption scheme, and he sends the resulting ciphertext back to Alice. Notice that the decryption of the latter ciphertext together with σ uniquely determine $x \cdot y$ (and it is enough for Alice and Bob to output $xy + \sigma$ and $-\sigma$ respectively to obtain the desired shares).

Our Contributions. The purpose of this note is to highlight certain subtle points when instantiating the above process to obtain malicious security. Specifically, motivated by applications to threshold-ECDSA, we look at a specific AHE-based multiplication protocol, i.e. instantiations of the aforementioned multiplication process under a specific regime of parameters, and we identify a number of vulnerabilities. Then, we propose a simple solution to completely remediate the issue.

Notation. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ denote two parties holding inputs x and y respectively such that $x, y \in [2^\ell]$ for some $\ell \in \mathbb{N}$. Further assume that $\mathcal{P}_1$ is associated with a secret-public key-pair $(\mathsf{sk}, \mathsf{pk})$ for some AHE. We further assume that the plaintext-space of the AHE is equal to $(\mathbb{Z}_N, +)$, for some RSA modulus N.[4] For correctness, it is assumed that N is much larger than 2^ℓ.

2PC Multiplication w/ AHE. Party $\mathcal{P}_1$ (the initiator) encrypts their input x under their own key. $\mathcal{P}_1$ sends the resulting ciphertext $C = \mathsf{enc}_{\mathsf{pk}}(x)$ to $\mathcal{P}_2$ (the responder), together with a *proof* that the plaintext value of C lies in the range $[0, 2^\ell]$. Then, $\mathcal{P}_2$ samples $\sigma \leftarrow \{0, 2^\lambda\}$ and calculates $D = \mathsf{enc}_{\mathsf{pk}}(xy + \sigma)$, using the homomorphic properties of the encryption scheme. $\mathcal{P}_2$ sends D, together with a *proof* that ciphertext D was obtained as an affine-like operation on C such that the multiplicative coefficient (i.e. y) lies in range $[0, 2^\ell]$ and the additive coefficient (i.e. s_2) lies in range $[0, 2^\lambda]$; the accompanying ZK-proofs are typically referred to as *range proofs*. In the end, $\mathcal{P}_2$ outputs $s_2 = -\sigma$ and $\mathcal{P}_1$ outputs $s_1 = \mathsf{dec}_{\mathsf{sk}}(D)$. Observe that, in an honest execution, $s_1 + s_2 = x \cdot y$ *over the integers* $\mathbb{Z}$. For certain regime of parameters, we show how the protocol can be attacked *sometimes*.

Regime of Parameters. As per [3, 4], we have that $2^\lambda \approx N$ (i.e. the size of the plaintext-space of the encryption scheme). Observe that in this regime of parameters σ can be drawn from the entire domain of $\mathbb{Z}_N$, and recall that 2^ℓ is much smaller than N. For concrete parameters, one can think of $N \approx 2^{8 \cdot \ell}$. In the sequel, we show that a malicious responder $\mathcal{P}_2$ can choose a suitable σ such that $s_1 + s_2 = xy$ over $\mathbb{Z}$ if and only if the i most significant bits of x are zero[5], for any (say) constant i. We refer to this attack as the *bit-probing attack.*

Further Attacks on Cheaper Variant of the Protocol. When the proofs are altogether omitted, we show further attacks in Section 4.2, including an attack that can potentially reveal the entire secret when the attack is mounted iteratively.

Bit-Probing Attack. Let $x_0 \dots x_{\ell-1}$ denote the bit decomposition of x and notice that the i most significant bits are zero if and only $x < 2^{\ell-i}$. Therefore, the attacker ($\mathcal{P}_2$) achieves their goal by choosing $y = 1$ and $\sigma = N - 2^{\ell-i}$, and thus $s_1 + s_2 = x$ if and only if the i most significant bits of x are all-zero. In turn, when used within a different protocol (e.g. threshold ECDSA), the bit-probing attack gives rise to a selective-failure which results in a biased output (biased ECDSA signatures are very vulnerable)

3 Preliminaries

Definition 3.1. We say that $N \in \mathbb{N}$ is a Paillier-Blum integer iff $\gcd(N, \phi(N)) = 1$ and $N = pq$ where p, q are primes such that $p, q \equiv 3 \mod 4$.

Definition 3.2 (Paillier Encryption)**.** Define the Paillier cryptosystem as the three tuple $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ below.

1. Let $(N; p, q) \leftarrow \mathsf{gen}(1^\kappa)$ where p and q are $\kappa/2$-long primes and $N = pq$. Write $\mathsf{pk} = N$ and $\mathsf{sk} = (p, q)$.
2. For $m \in \mathbb{Z}_N$, let $\mathsf{enc}_{\mathsf{pk}}(m; \rho) = (1 + N)^m \cdot \rho^N \mod N^2$, where $\rho \leftarrow \mathbb{Z}_N^*$.
3. For $c \in \mathbb{Z}_{N^2}$, letting $\mu = \phi(N)^{-1} \mod N$,

$$\mathsf{dec}_{\mathsf{sk}}(c) = \left(\frac{[c^{\phi(N)} \mod N^2] - 1}{N} \right) \cdot \mu \mod N.$$

[4]We inherit this convention from the Paillier cryptosytem. Our observations hold for many other finite structures.
[5]This attack can be modified for any value of the most significant bits

Definition 3.3 (Pedersen Commitments)**.** Define Pedersen as the three-tuple $(\mathsf{gen}, \mathsf{com}, \mathsf{dcm})$ below.

1. Let $(N, s, t; p, q) \leftarrow \mathsf{gen}(1^\kappa)$ where p and q are $\kappa/2$-long strong primes and $N = pq$, number t is a random quadratic residue in $\mathbb{Z}_N^*$ and $s \leftarrow \langle t \rangle$.
2. For $m \in \mathbb{Z}$, let $\mathsf{com}(m; \rho) = C = s^m t^\rho \mod N$, where $\rho \leftarrow \mathbb{Z}_N$.
3. For $C \in \mathbb{Z}_N^*$, and $(m, \rho) \in \mathbb{Z} \times \mathbb{Z}_N$, $\mathsf{dcm}(C, m, \rho) = 1$ if and only if $C = s^m t^\rho \mod N$.

Paillier Encryption in Range. For Paillier public key N, the following relation verifies that the plaintext value of Paillier ciphertext C lies in range $\mathcal{I}$. Define

$$R_{\mathsf{enc}} = \left\{ (N, \mathcal{I}, C, ; x, \rho) \mid x \in \mathcal{I} \ \wedge \ C = (1+N)^x \rho^N \in \mathbb{Z}_{N^2}^* \right\}.$$

Paillier Affine Operation in Range. For Paillier public key N, the following relation verifies that a Paillier ciphertext $D \in \mathbb{Z}_{N^2}^*$ was obtained as an affine-like transformation on C such that the multiplicative coefficient (i.e. α) lies in the range $\mathcal{I}$, and the additive coefficient (i.e. β) lies in the the range $\mathcal{J}$. Define

$$R_{\mathsf{aff}} = \{ (N, \mathcal{I}, \mathcal{J}, D, C; \alpha, \beta, \rho) \mid (\varepsilon, \delta) \in \mathcal{I} \times \mathcal{J} \ \wedge \ D = C^\alpha \cdot (1+N)^\beta \rho^N \in \mathbb{Z}_{N^2}^* \}$$

Definition 3.4. A NIZK proof system $\Pi_{\Pi_{\mathsf{nizk}}}^{R,\mathcal{H}}$ for relation R is a tuple $(\mathsf{P}^\mathcal{H}, \mathsf{V}^\mathcal{H})$ of PPT algorithms with access to random oracle $\mathcal{H}$ such that

- $\mathsf{P}^\mathcal{H}$ takes input (x, w) and outputs a string ψ.
- $\mathsf{V}^\mathcal{H}$ takes input (x, ψ) and outputs a bit b.
  - *Completeness.* If $(x, w) \in R$ then $\mathsf{V}(x, \psi) = 1$, with overwhelming probability over $\psi \leftarrow \mathsf{P}^\mathcal{H}(x, w)$.
  - *Soundness.* If x is false with respect to R (i.e $(x, w) \notin R$ for all w), then for any PPT algorithm $\mathsf{P}^{*\mathcal{H}}$, then $\mathsf{V}^\mathcal{H}(x, \psi^*) = 0$ with overwhelming probability over $\psi^* \leftarrow \mathsf{P}^{*\mathcal{H}}(x)$.
  - *HVZK.* There exists a simulator $\mathcal{S}$ and oracle $\hat{\mathcal{H}}$ such that for $\hat{\psi} \leftarrow \mathcal{S}(x)$, it holds and $\mathsf{V}^{\hat{H}}(x, \hat{\psi}) = 1$ for every x, with overwhelming probability over the random coins of $\mathcal{S}$. Furthermore, the following distributions are statistically indistinguishable. For $(x, w) \in R$: $\psi \leftarrow \mathsf{P}^\mathcal{H}(x, w)$ and $\psi \leftarrow \mathcal{S}(x)$.

3.1 2PC Multiplication w/ Paillier

FIGURE 1 (2PC Multiplication)

- **Inputs:** Common input is N. $\mathcal{P}_1$ has secret input (x_1, p, q) and $\mathcal{P}_2$ has secret input x_2 s.t. $x_1, x_2 \in \pm 2^\ell$, $N = p \cdot q$ for $p, q > 2^\ell$.

1. Party $\mathcal{P}_1$ calculates $C = \mathsf{enc}_1(x_1, \rho)$ i.e. a fresh encryption of x_1 and $\psi \leftarrow \Pi_{\mathsf{nizk}}^{\mathsf{enc},\mathcal{H}}(\texttt{prove}, C, N; x_1, \rho)$.
   - $\mathcal{P}_1$ sends (C, ψ) to $\mathcal{P}_2$.
2. When obtaining (C, ψ), party $\mathcal{P}_2$ checks $\Pi_{\mathsf{nizk}}^{\mathsf{enc},\mathcal{H}}(\texttt{verify}, C, N, \psi) = 1$ and does:
   (a) Sample $\sigma \leftarrow \pm 2^\lambda$ and $\mu \leftarrow \mathbb{Z}_N$ set $D = C^{x_2} \cdot \mathsf{enc}_1(\sigma; \mu) \mod N^2$.
   (b) Calculate $\psi' \leftarrow \Pi_{\mathsf{nizk}}^{\mathsf{aff},\mathcal{H}}(\texttt{prove}, C, D, N; x_2, s_2, \mu)$.
   $\mathcal{P}_2$ sends (D, ψ') to $\mathcal{P}_1$.
3. When obtaining (D, ψ'), party $\mathcal{P}_1$ checks $\Pi_{\mathsf{nizk}}^{\mathsf{aff},\mathcal{H}}(\texttt{verify}, C, D, N, \psi') = 1$.

- **Outputs**: $\mathcal{P}_2$ outputs $s_2 = -\sigma$, $\mathcal{P}_1$ outputs $s_1 = \mathsf{dec}_1(D)$.

Figure 1: 2PC Multiplication

4 Technical Overview

4.1 Bit-Probing Attack

The first attack is very simple. The attacker $\mathcal{P}_2^*$ causes the plaintext to wrap around the modulus if and only if the most significant bits are all zero. This is achieved by choosing σ (the additive noise) to be equal to $N - 2^{\ell-i}$ and using value 1 as input for multiplication. See Figure 2 for the formal description.

FIGURE 2 (Attacker $\mathcal{P}_2^*$ – Bit-Probing Attack)

- **Inputs:** Common input is N. $\mathcal{P}_1$ has secret input (x, p, q) and s.t. $x \in \pm 2^\ell$, $N = p \cdot q$ for $p, q > 2^\ell$. $\mathcal{P}_2^*$ has input $i \in \mathbb{N}$.

1. Party $\mathcal{P}_1$ performs the same operations as in Figure 1.
2. When obtaining (C, ψ), party $\mathcal{P}_2^*$ does:
   (a) Sample $\mu \leftarrow \mathbb{Z}_N$ and set $D = C \cdot \mathsf{enc}(N - 2^{\ell-i}; \mu) \mod N^2$.
   (b) Calculate $\psi' \leftarrow \Pi_{\mathsf{nizk}}^{\mathsf{aff},\mathcal{H}}(\mathtt{prove}, C, D, N; y, s_2, \mu)$.
   $\mathcal{P}_2^*$ sends (D, ψ') to $\mathcal{P}_1$.
3. When obtaining (D, ψ'), party $\mathcal{P}_1$ checks $\Pi_{\mathsf{nizk}}^{\mathsf{aff},\mathcal{H}}(\mathtt{verify}, C, D, N, \psi') = 1$.

- **Outputs**: $\mathcal{P}_2^*$ outputs $s_2 = -N + 2^{\ell+i}$, $\mathcal{P}_1$ outputs $s_1 = \mathsf{dec}(D)$.

Figure 2: Attacker $\mathcal{P}_2^*$ – Bit-Probing Attack

Claim 4.1. *It holds that $s_1 + s_2 = x$ iff the i most significant bits of x are all zero.*

Proof. Write $x = \sum_{j=0}^{\ell-1} x_j 2^j$. The i most significant bits of x are all zero iff $x < 2^{\ell-i}$. Therefore, if $x < 2^{\ell-i}$ then $s_1 = x + N - 2^{\ell-1}$ and, conversely, if $x \geq 2^{\ell-i}$, then $s_1 = x - 2^{\ell-1}$, and the claims follows. □

Remark 4.2. The above can be extended to any guess of the bits (not just all-0). Write $g = \sum_{j=\ell-i}^{255} g_i \cdot 2^j$ for the adversary's guess. Modify the above attack by setting $D = C \cdot (1+N)^{-g+N-2^{\ell-i}} \mod N^2$ and $s_2 = -N + g - 2^{\ell-i} \mod q$.

4.1.1 How to Remediate the Attack.

One immediate way to solve this issue is to move to the second regime of parameters using a valid (non-faulty) ZK-proof. Alternatively, we also propose a different fix which requires minimal changes to existing systems (though it is important to make sure that the range-proofs on the parties' inputs are sound, otherwise a combination of the bit-probing and hidden-giant attack is still possible (c.f. Section 4.2). We propose the following check performed by $\mathcal{P}_1$ after decrypting $s_1 = \mathsf{enc}_{\mathsf{sk}}(D)$: $\mathcal{P}_1$ samples a random $\varepsilon \leftarrow [0, \sqrt{N}]$ and reports a failure if $s_1 \notin [\varepsilon, N - \varepsilon]$, or proceeds like before and simply outputs s_1 if not.

4.2 Attacks when ZK-Proofs are Ommited

There are several additional attacks that can be carried out when the range proofs are entirely omitted. Next we present how a corrupted $\mathcal{P}_2$ can guess an arbitrary number of the honest party's least significant bits, or an arbitrary combination of the most significant and least significant. In addition, in Section 4.2.3, we show how a malicious $\mathcal{P}_1$ may attempt to guess $\mathcal{P}_2$'s input bit by bit (i.e. when $\mathcal{P}_2$ uses the same input repeatedly in different executions).

In the sequel, it is assumed that the parties perform the multiplication modulo a prime q_0 (unrelated to the factorization and much smaller than the Paillier modulus N).

FIGURE 3 (Attacker $\mathcal{P}_2^*$ Probing LSB's)

- **Inputs:** Common input is N. $\mathcal{P}_1$ has secret input (x, p, q) and s.t. $x \in \pm 2^\ell$, $N = p \cdot q$ for $p, q > 2^\ell$. $\mathcal{P}_2^*$ has input $i \in \mathbb{N}$.

1. Party $\mathcal{P}_1$ performs the same operations as in Figure 1.
2. When obtaining (C, ψ), party $\mathcal{P}_2^*$ does:

   Set $u = 2^{-i} \mod N$ and $D = C^u \mod N^2$, and sends D to $\mathcal{P}_1$.

- **Outputs**: $\mathcal{P}_2^*$ outputs 0, $\mathcal{P}_1$ outputs $s_1 = \mathsf{dec}_1(D) \mod q_0$.

Figure 3: Attacker $\mathcal{P}_2^*$ Probing LSB's

4.2.1 Bit-Probing for Least Significant Bits

Claim 4.3. *It holds that $s_1 + s_2 = 2^{-i}x \mod q_0$ iff the i least significant bits of x are all zero.*

Proof. Define the Bézout coefficients (u, v) and (λ, μ) such that $u \cdot 2^i + v \cdot N = 1$ and $\lambda \cdot 2^i + \mu \cdot q_0 = 1$ and notice that $|\mu| \leq 2^i$. Further notice that $s_1 + s_2 = 2^{-i}x \mod q_0$ is equivalent to the existence of $\alpha < N/q_0$ such that $(\lambda - u)x = \alpha \cdot q_0 \mod N$. Calculate

$$\begin{aligned}
(\lambda - u)x &= \alpha \cdot q_0 \mod N && \Longleftrightarrow \\
\left(\frac{1 - \mu \cdot q_0}{2^i} - u\right) x &= \alpha \cdot q_0 \mod N && \Longleftrightarrow \\
\left((1 - \mu \cdot q_0) - 2^i u\right) x &= 2^i \cdot \alpha \cdot q_0 \mod N && \Longleftrightarrow \\
((1 - \mu \cdot q_0) - 1)\, k &= 2^i \cdot \alpha \cdot q_0 \mod N && \Longleftrightarrow \\
\mu \cdot k &= 2^i \cdot \alpha \mod N
\end{aligned}$$

Since neither side of the congruence wraps around N, and μ and 2^i are coprime, we deduce that 2^i divides k. ☐

Remark 4.4. The above can be extended to any guess of the bits (not just all-0). Write $g = \sum_{j=0}^{i-1} g_j \cdot 2^j$ for the adversary's guess. Modify the above attack by setting $D = (C \cdot (1 + N)^{-g})^u \mod N^2$ and $s_2 = g \cdot 2^{-i} \mod q_0$.

4.2.2 Bit-Probing for MSB's & LSB's.

FIGURE 4 (Attacker $\mathcal{P}_2^*$ Probing LSB's & MSB's)

- **Inputs:** Common input is N. $\mathcal{P}_1$ has secret input (x, p, q) and s.t. $x \in \pm 2^\ell$, $N = p \cdot q$ for $p, q > 2^\ell$. $\mathcal{P}_2^*$ has inputs $i, j \in \mathbb{N}$.

1. Party $\mathcal{P}_1$ performs the same operations as in Figure 1.
2. When obtaining (C, ψ), party $\mathcal{P}_2^*$ does:

   Set $u = 2^{-i} \mod N$ and $D = C^u \cdot (1 + N)^{N - 2^\ell - j - i} \mod N^2$, and sends D to $\mathcal{P}_1$.

- **Outputs**: $\mathcal{P}_2^*$ outputs $N - 2^\ell - j - i \mod q_0$, $\mathcal{P}_1$ outputs $s_1 = \mathsf{dec}_1(D) \mod q_0$.

Figure 4: Attacker $\mathcal{P}_2^*$ Probing LSB's & MSB's

Claim 4.5. *It holds that $s_1 + s_2 = 2^{-i}x \mod q_0$ iff the i least significant bits and the j most significant bits of x are all zero.*

Proof. Assume that to 2^i does not divide x. Let $ux = \sigma + Ns$ for $\sigma \in [N]$, and, depending on $\sigma > 2^{\ell-i-j}$, either $s_1 = (ux - Ns) + N - 2^{\ell-i-j}$ or, $s_1 = (ux - Ns) - 2^{\ell-i-j}$. Therefore, if $s_1 + s_2 = 2^{-i}x \mod q_0$, then either $(ux - Ns) + N = \lambda x + r \cdot q_0$ or $(ux - Ns) + N = \lambda x + r \cdot q_0$ with $r < N/q_0$. In either case, you get that $(u - \lambda)x \mod N = rq_0$ which boils down to 2^i divides x – contradiction. Next, assume that 2^i divides x and deduce that the test passes only if the most significant bits of x are zero. □

Remark 4.6. For an arbitrary guess just remove the guess from C and adjust s_2 accordingly.

4.2.3 Guessing $\mathcal{P}_2$'s input bit by bit.

FIGURE 5 (Attacker $\mathcal{P}_1^*$ guessing $\mathcal{P}_2$'s input bit by bit)

- **Inputs:** Common input is N. $\mathcal{P}_2$ has secret input (x, p, q) and auxiliary input $y^* = y \mod 2^j$. $\mathcal{P}_2$ has input y. Write $v = 2^{-j-1} \mod N$ and $\gamma = 2^{-j-1} \mod q_0$.

1. Party $\mathcal{P}_1$ sends $C = \mathsf{enc}(v)$ to $\mathcal{P}_1$.
2. When obtaining C, party $\mathcal{P}_2^*$ performs the same operation as in Figure 1.

- **Outputs**: $\mathcal{P}_1^*$ outputs $s_1 = [\mathsf{dec}(D) - vy^* \mod N] + \gamma y^* \mod q_0$, $\mathcal{P}_2$ outputs s_2.

Figure 5: Attacker $\mathcal{P}_1^*$ guessing $\mathcal{P}_2$'s input bit by bit

Claim 4.7. *It holds that $s_1 + s_2 = 2^{-j-1}x \mod q_0$ iff $y^* = y \mod 2^{j+1}$, i.e. if the $(j+1)$-th bit of y is* 0.

Proof. Write β for the noise added by $\mathcal{P}_2$ and notice that

$$[v \cdot (y_j 2^j + y_{j+1} 2^{j+1} + \ldots) + \beta \mod N] + \gamma(y_0 + \ldots + y_{j-1} 2^{j-1})$$
$$= \begin{cases} 2^{-(j+1)} \cdot y + \beta \mod q_0 & \text{if } y_j = 0 \\ [2^{-1} \mod N] + \beta + 2^{-(j+1)} \cdot y \mod q_0 & \text{if } y_j = 1 \ \wedge \ \neg\mathsf{wrap} \\ [2^{-1} \mod N] + \beta - N + 2^{-(j+1)} \cdot y \mod q_0 & \text{if } y_j = 1 \ \wedge \ \mathsf{wrap} \end{cases}$$

Conclude that $s_1 + s_2 = 2^{-(j+1)}y \mod q_0$ if and only if $y_j = 0$, since $(N+1)/2$ and $(-N+1)/2 \neq 0 \mod q_0$. □

References

[1] D. Boneh, S. Halevi, and N. Howgrave-Graham. The modular inversion hidden number problem. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2001. doi: 10.1007/3-540-45682-1_3.

[2] R. Canetti, N. Makriyannis, and U. Peled. Uc non-interactive, proactive, threshold ecdsa. Cryptology ePrint Archive, Report 2020/492, 2020. https://eprint.iacr.org/2020/492.

[3] R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1179–1194, 2018. doi: 10.1145/3243734.3243859. URL https://doi.org/10.1145/3243734.3243859.

[4] R. Gennaro and S. Goldfeder. One round threshold ECDSA with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020:540, 2020. URL https://eprint.iacr.org/2020/540.

[5] J. Xu, S. Sarkar, L. Hu, H. Wang, and Y. Pan. New results on modular inversion hidden number problem and inversive congruential generator. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 297–321, Cham, 2019. Springer International Publishing. ISBN 978-3-030-26948-7.