Multi-Target Attacks on the Picnic Signature Scheme and Related Protocols

Itai Dinur[1] and Niv Nadler[2]

[1] Department of Computer Science, Ben-Gurion University, Israel
[2] Independent

Abstract. Picnic is a signature scheme that was presented at ACM CCS 2017 by Chase et al. and submitted to NIST's post-quantum standardization project. Among all submissions to NIST's project, Picnic is one of the most innovative, making use of recent progress in construction of practically efficient zero-knowledge (ZK) protocols for general circuits.
In this paper, we devise multi-target attacks on Picnic and its underlying ZK protocol, ZKB++. Given access to S signatures, produced by a single or by several users, our attack can (information theoretically) recover the κ-bit signing key of a user in complexity of about $2^{\kappa-7}/S$. This is faster than Picnic's claimed 2^{κ} security against classical (non-quantum) attacks by a factor of $2^7 \cdot S$ (as each signature contains about 2^7 potential attack targets).
Whereas in most multi-target attacks, the attacker can easily sort and match the available targets, this is not the case in our attack on Picnic, as different bits of information are available for each target. Consequently, it is challenging to reach the information theoretic complexity in a computational model, and we had to perform cryptanalytic optimizations by carefully analyzing ZKB++ and its underlying circuit. Our best attack for $\kappa = 128$ has time complexity of $T = 2^{77}$ for $S = 2^{64}$. Alternatively, we can reach the information theoretic complexity of $T = 2^{64}$ for $S = 2^{57}$, given that all signatures are produced with the same signing key.
Our attack exploits a weakness in the way that the Picnic signing algorithm uses a pseudo-random generator. The attack is mitigated in the recent Picnic 2.0 version.
In addition to our attack on Picnic, we show that a recently proposed improvement of the ZKB++ protocol (due to Katz, Kolesnikov and Wang) is vulnerable to a similar multi-target attack.

Keywords: Cryptanalysis, multi-target attack, Picnic, signature scheme, zero-knowledge protocol, ZKB++, MPC, block cipher, LowMC.

1 Introduction

Multi-target attacks are among the most basic attacks against cryptosystems that are built using symmetric-key primitives. In a typical example, the attacker first obtains G possible *targets*, which correspond to outputs of the cryptosystem, evaluated with different secret keys (or secret inputs, in general). Then, the

attacker guesses a key, evaluates the cryptosystem, and compares the result with all targets. Based on a standard birthday paradox argument, the expected workload of the attacker for hitting one of the targets is reduced by a factor of (at least[1]) G, compared to the workload of hitting a single target.[2]

In our multi-target attack model, we deal with a cryptosystem with U users, each with a long-term key. For each user $i \in [1, U]$, the attacker obtains D_i data points created by this user and we denote $D = \sum_{i=1}^{U} D_i$. Each data point may be additionally associated with a short-term key. The goal of the attacker is to recover one of the keys for the cryptosystem (either a short or a long-term key). For example, in a signature scheme, each user has a long-term signing key, a data point may be a signature and a short-term key is (secret) randomness used in creating the signature. We note that in many cases the recovery of a short-term key allows recovering the corresponding user's long-term key, but this possibility is not directly captured by our simple model.

We distinguish between three types of multi-target attacks according to the number of targets G they present to an attacker.[3]

1. *Multi-user single-target attack*: G is determined by the number of users U, i.e., $G = U$. Typically, this occurs if the long-term user keys are vulnerable to a multi-target attack.
2. *Single-user multi-target attack*: G is determined separately for each user as $G_i = D_i$. Hence, the best attack uses $G = \text{argmax}_i\{D_i\}$. In this case, the short-term keys of each user are vulnerable to a multi-target attack.
3. *Multi-user multi-target attack* (or generic multi-target attack): G is determined by the total number of available data points D, i.e., $G = D$. Here, all short-term keys are vulnerable to a multi-target attack. In principle, this is the most powerful type of multi-target attack, as all data points can simultaneously be used by the attacker as targets.

A standard way to mitigate multi-target attacks is to add a public random input to the cryptosystem (i.e., a salt), thus creating a different tweaked variant of it per salt. Since one has to choose a particular salt in order to evaluate the cryptosystem with a secret key, salting forces the attacker to focus on only one target per secret key guess.

In this paper, we are mainly interested in public key cryptosystems that are based on symmetric-key primitives. These cryptosystems have received significant attention recently due to their alleged post-quantum security. The most well-known category within this class consists of hash-based signatures, which originate from Lamport's one-time signatures [14]. In recent years, these signatures have been subject to many optimizations and improvements until the recent development of practical stateless hash-based signatures [3]. As all cryptosystems built with symmetric-key primitives, hash-based signature are poten-

[1] If the keys are not generated uniformly, the workload of the attack could be lower.

[2] Throughout this paper, we focus on attacks run on classical computers, but our analysis can be extended to deal with attacks on quantum computers.

[3] Our model is related to the one of [11], but our classification is at a higher level.

tially vulnerable to multi-target attacks and substantial effort has been put into their efficient mitigation (cf. [11]).

Another public key cryptosystem that is based on symmetric-key components is the Picnic signature scheme. It was presented at ACM CCS 2017 [6] by Chase et al. and submitted [5] to NIST's post-quantum standardization project [19].[4] Picnic's design is solely based on symmetric-key primitives, yet is completely different from the design of hash-based signatures. Our main goal in this paper is to investigate the resistance of Picnic against multi-target attacks. As we demonstrate, this requires dedicated analysis due to Picnic's novel design. We note that our description of Picnic and its analysis applies to Picnic 1.0 and not to the recent Picnic 2.0 version [5].

Picnic The Picnic signature scheme uses the ZKB++ zero-knowledge (ZK) protocol (that improves upon the original ZKBoo protocol [10] in terms of efficiency), which allows to non-interactively prove knowledge of a preimage x to a public value y under a one-way function f. In Picnic, y is part of the public key, whereas x is the secret signing key. In order to sign a message, the signer uses ZKB++ to prove knowledge of x, where the message is embedded in the signing process to generate (pseudo) random bits.

The ZKB++ protocol employs the "MPC-in-the-head" paradigm due to Ishai et al. [12]. In order to prove knowledge of x, the prover (signer), simulates a multi-party computation (MPC) protocol between several players (whose number is 3 in ZKB++) that receive shares of x and compute $f(x) = y$. The prover then commits to the different internal states (views) of each of the players, and the verifier challenges the prover by asking to open the commitments of a subset of the players, revealing their views.

The *correctness* of the MPC protocol guarantees that if the prover does not know x and tries to cheat, then the joint views of some of the players are inconsistent. Hence, the verifier can catch a cheating prover with some probability, which is amplified by repeating the process. The *privacy* guarantee of the MPC protocol ensures that opening the views of a (sufficiently small) subset of players does not reveal any information about x, hence the secret signing key is not leaked. The proof is made non-interactive using the Fiat-Shamir transform [9]. More specifically, the prover computes the challenge by hashing the commitments, where in Picnic, the message to be signed is hashed as well (making the signature depend on the message).

A Picnic signature thus comprises of partial transcripts of several independent runs of the MPC protocol, where for each run, the views of two out of three participating (virtual) players are opened. As noted above, a view contains the

[4] The ACM CCS 2017 paper [6] introduced two signature scheme variants: Fish (which uses the Fiat-Shamir transform [9]), with claimed security against classical computer attacks, and Picnic (which uses Unruh's transform [21]), with claimed security against quantum computer attacks. In the NIST submission [5], these variants were renamed to Picnic-FS and Picnic-UR, respectively. Our analysis applies to both variants, but we focus on Picnic-FS for simplicity.

player's internal states computed during the MPC protocol. The signature also includes the player's sampled random bits, so that the view's consistency can be checked by a verifier of the signature. However, having the signature include all the random bits sampled by the "opened players" blows up its size. Hence, Picnic uses a standard optimization, where each player only samples a short seed of size κ bits (where κ is the security level against classical attacks), and produces the random bits required by the protocol using a deterministic pseudo-random generator (PRG), initialized with the seed. Thus, the short seeds of the opened players are included in the signature for each run and the verifier uses them to compute the required pseudo-random bits. Obviously, the random seed (and view) of the remaining "unopened player" in each run must not be included in the signature, as it may expose the secret key x.

Multi-Target Attacks on Picnic Our main result is a multi-target attack on Picnic. The first step of the attack involves collecting signatures (produced by one or several users) containing (partial) transcripts of various runs of the MPC protocol. Then, by independently guessing a value of the κ-bit seed and evaluating the PRG, the attacker can match and detect that the seed is used by the unopened player in a particular run. Once the seed of the unopened player in a run is revealed, the secret signing key of the corresponding user can be computed easily. Thus, given a total of D runs, the attacker needs to test an average of $2^\kappa/D$ seeds until a match with a run is detected. The attack is thus a generic multi-target attack (i.e., a multi-user multi-target attack) and it violates Picnic's claims of κ-bit security (against attacks by classical computers).

A crucial detail missing from the attack's outline above is how to detect a match between a guessed seed and the seed used by an unopened player in an available run. In fact, this may seem impossible, as the privacy of the MPC protocol should presumably prevent the pseudo-random bits used by the unopened player from leaking. This issue is related to a subtlety about MPC protocols: their privacy guarantees apply to the input of each player and not (necessarily) to the (pseudo) random bits that each player uses. In other words, MPC protocols are allowed to (and mostly do) expose some (pseudo) random bits used by each player and still remain private, i.e., protect the players' inputs. On the other hand, it is generally important that not all of a player's randomness is exposed, as this leaks the player's input. In the context of Picnic, in each run, some output bits of the PRG used by the unopened player can be easily computed by the attacker, which makes it possible to detect that the unopened player uses a certain seed once it is correctly guessed (and then compute the secret key).

We note that the Picnic designers attempted to protect it against multi-target attacks. For example, the public key of each owner i defines a different one-way function f_i,[5] rather than having all owners prove knowledge of a preimage under the same function f. Indeed, a global choice of f allows the attacker to

[5] Internally, Picnic uses a block cipher encryption $f_i(x) = \mathrm{Enc}_x(p(i))$, where a different plaintext $p = p(i)$ is used for each public key owner (defining a different encryption function).

mount a multi-user single-target attack by computing a preimage to one out of many images available in the different public keys. Yet, Picnic was not protected against our generic (and more powerful) multi-user multi-target attack against the seeds, presumably because it is not obvious that such an attack is possible (as previously mentioned). Internally, the security proof of Picnic (published in its design document [5]) simply does not consider attacker queries with arbitrary seed values to the PRG and hence does not cover our attack.

Randomness Extraction According to the birthday paradox, the expected complexity of our attack is $T = 2^\kappa/D$ (for $D \leq 2^{\kappa/2}$). However, this information theoretic analysis assumes that the attacker wins once the PRG is evaluated with a seed that is used by the unopened player in one of the available runs (as enough information is available to recover the key). In practice, achieving the information theoretic complexity is challenging, since the PRG output bits of the unopened player that can be computed in each run, vary according to the run. Therefore, a standard matching algorithm which sorts the runs according to the available PRG output bits does not work, while its naive extension has very high complexity (e.g., at least 2^{102} for $\kappa = 128$). Consequently, we carefully analyze Picnic (and its underlying block cipher LowMC that implements f [1]) in order to extract the maximal amount of PRG output data from each run. We then utilize this data by devising a dedicated attack algorithm that that recovers the signing key and outperforms the naive algorithm by a factor of up to 2^{25} for $\kappa = 128$ and by more than 2^{30} for larger κ values.

These techniques we use for extracting the maximal amount of PRG output data mainly involve exploiting dependencies among private values computed by a player and masked with PRG output bits. As a simple example, assume that a player outputs 3 bits z_1, z_2, z_3 such that $z_1 = v_1 \cdot v_2 \oplus r_1$, $z_2 = v_2 \cdot v_3 \oplus r_2$ and $z_3 = v_1 \cdot v_3 \oplus r_3$ where v_1, v_2, v_3 are internal private bit values and r_1, r_2, r_3 are PRG output bits. Observe that the triplet $v_1 \cdot v_2, v_2 \cdot v_3, v_1 \cdot v_3$ can only attain 5 values (as the values $011, 101, 110$ as impossible). Hence, given z_1, z_2, z_3, the triplet of bits r_1, r_2, r_3 can only attain 5 out of 8 possible values, revealing information about them.

Although our techniques are tailored to the Picnic circuit, they can be easily adapted and applied to other MPC protocols in order to extract information about the random bits that are used by the players. Such extraction techniques may be relevant to attackers is scenarios that extend beyond multi-target attacks. For example, the attacker's ability to exploit a weak PRG in a cryptographic protocol (e.g., by predicting its output) may depend on the number of PRG output bits available. This was demonstrated in [7] by Checkoway et al.[6] which investigated the exploitability of the Dual EC weak PRG in TLS implementations. Additionally, in case protocol implementations generate seeds with low entropy, extraction techniques may allow an attacker to efficiently detect that two protocol executions use the same seed and to violate their security. To the

[6] We thank an anonymous reviewer for pointing out the link between [7] and our paper.

best of our knowledge, this is the first paper that investigates such randomness extraction techniques for MPC protocols.

Concrete Complexity of the Main Attack In terms of concrete complexity, we are interested in attacks that utilize at most 2^{64} signatures. This is the limit set in NIST's Call for Proposals document [19] on the number of signatures produced per signing key.[7] For Picnic, each signature contains $R \in \{219, 324, 438\}$ runs depending on the desired security level against classical attacks, $\kappa \in \{128, 192, 256\}$, respectively. The complexities of our main attack for each desired security level are summarized below.

- For $\kappa = 128$, we can reach the information theoretic complexity of $T = 2^{\kappa}/D$ up to $D = 2^{42}$ (using about 2^{35} signatures), and obtain $T \approx 2^{128-42} = 2^{86}$. When 2^{64} signatures are available, we can recover a secret signing key with complexity of about 2^{77}.
- For $\kappa = 192$, we achieve the information theoretic complexity $T = 2^{\kappa}/D$ for almost all $D \leq 324 \cdot 2^{64} \approx 2^{72}$. The best complexity is $T = 2^{124}$, obtained for $D = 2^{72}$.
- For $\kappa = 256$, we achieve the information theoretic complexity $T = 2^{\kappa}/D$ for all $D \leq 438 \cdot 2^{64}$.

Seed Collision Attack Interestingly, for $\kappa = 128$, we can reach the information theoretic complexity for the specific case of $D = 2^{64}$ (i.e. utilizing about 2^{57} signatures) using another attack, given that all the available signatures are produced with the same signing key.[8] While the attack resembles a single-user multi-target attack, it is not a classical multi-target attack in the sense that the attacker does not guess any key material (such as PRG seeds). Instead, the attacker waits for a specific *seed collision* event (in which two different runs use the same PRG seed for the unopened player) to occur on the observed data. Once the event is detected, the user's signing key can be efficiently recovered using the known PRG output bits of the unopened player in both runs. The attack can be extended (with a limited range of parameters) to recover the signing key of one out of many users, e.g., if the attacker collects 2^{64} signatures ($D \approx 2^{71}$), produced with up to 2^{14} private keys.

Multi-Target Attacks on Additional Cryptosystems Our multi-target attack is, in fact, an attack on the ZKB++ protocol, as well as the previous ZKBoo protocol. Therefore, the attack also carries over to additional cryptosystems that were built using these protocols. This includes the ring-signature and additional constructions of [4, 8], whose implementations are based in ZKB++.

[7] As the attacker may acquire signatures produced with various signing keys, our model is somewhat more restrictive than NIST's.

[8] The attack can also be applied to $\kappa \in \{192, 256\}$, but it requires significantly more than 2^{64} signatures.

We further analyze in this paper a recently proposed protocol due to Katz, Kolesnikov and Wang [13] (KKW), which was presented at ACM CCS 2018. The KKW protocol describes a new way to instantiate the MPC-in-the-head approach, yielding shorter proofs compared to ZKB++. Interestingly, the KKW protocol is vulnerable to a multi-target attack which is similar to our main attack on Picnic (and ZKB++).

In the penultimate section of the paper, we describe multi-target attacks on additional cryptosystems, which are made possible due to several design optimizations (mostly for MPC protocols). Unlike the case of Picnic, these multi-target attacks are standard and their descriptions only requires a very high-level understanding of the cryptosystems. Yet, the aim of this section is to show that some common optimizations do not come without a cost, which needs to be considered in cryptosystems that are designed for practical use.

Picnic 2.0 We notified the Picnic designers about the attack and they confirmed our findings. The weakness is addressed in the Picnic 2.0 version [5] by appending an additional salt to each signature. The salt is carefully used in generating the pseudo-random bits of each player in each run, such that multi-target attacks are mitigated. We further note that Picnic 2.0 added additional instances that use the KKW protocol, while in this paper we describe and analyze Picnic 1.0 which only uses ZKB++.

Paper Organization The rest of this paper is organized as follows. In Section 2 we describe Picnic and its building blocks, while in Section 3 we partially summarize the KKW protocol. Next, in Section 4, we outline the main steps of our multi-target attacks on Picnic and the KKW protocol. In Section 5, we elaborate on our main multi-target attack on Picnic, while our seed collision attack is described in Section 6. Finally, we describe multi-target attacks on additional cryptosystems in Section 7 and conclude in Section 8.

2 ZKBoo, ZKB++ and Picnic

ZKBoo is a ZK protocol described in [10]. An optimized variant of ZKBoo (called ZKB++) was later described in [6], which used it to construct the Picnic signature scheme. In this section, we give a brief overview of these constructions.

2.1 Overview of ZKBoo

The goal ZKBoo is to prove knowledge of a witness for a relation $Re := \{(x, y), f(x) = y\}$, where y is public and x is kept private. For example, given a 256-bit string y, we aim to prove knowledge of a preimage of y under SHA-256, namely, a string x such that $y = \text{SHA-256}(x)$.

ZKBoo employs the MPC-in-the-head paradigm of Ishai et al. [12], that we now outline very briefly. It uses some MPC protocol that implements f on input

shares of the secret witness x. The prover simulates the MPC protocol "in the head" and commits to the state and transcripts of all players. The verifier then "corrupts" a random subset of the simulated players by requesting to see their complete states. The verifier checks that the computation was done correctly from the perspective of the corrupted players, obtaining some assurance that the output is correct and the prover knows x. Iterating this procedure many times gives the verifier high assurance.

ZKBoo improves upon the practical efficiency of the MPC-in-the-head approach by replacing the MPC with a *circuit decomposition*, which does not necessarily need to satisfy classical MPC protocol properties. The circuit decompositions in ZKBoo involves 3 players. Given a circuit ϕ that computes f, it defines the following functions.

- Share: splits the input x into 3 shares.
- $\text{Output}_{i\in\{1,2,3\}}$: takes as input all of the input shares and some randomness and produces an output share for each of the players.
- Reconstruct: takes as input the three output shares and reconstructs the circuit's final output.

The circuit decomposition should satisfy the correctness property which means that its execution on input x must yield $f(x)$. It must further satisfy the 2-privacy property which requires that revealing the views (i.e., the values of the intermediate computation states) of any two players does not leak information about the witness x.

Given a circuit decomposition for ϕ, the ZKBoo protocol is a Σ-protocol for languages of the form $L := \{y \mid \exists x \,:\, y = \phi(x)\}$. As outlined below, it gives a non-interactive ZK proof of knowledge system for the relation using the Fiat-Shamir transform [9].

The computation $\phi(x)$ using the decomposition is a randomized algorithm called a *run*. As indicated above, in each run, each player $P_{i\in\{1,2,3\}}$ uses some (pseudo) random bits, generated by random seeds k_1, k_2, k_3, respectively. For a parameter R that determines the total number of runs, a proof is constructed as below.

1. For each run $i \in [1, R]$:
   (a) Sample $k_1^{(i)}, k_2^{(i)}, k_3^{(i)}$ and compute run i using the circuit decomposition outlined above.
   (b) For each player $P_1^{(i)}, P_2^{(i)}, P_3^{(i)}$, compute a commitment to its view during the run. The commitment for each player is computed by applying a hash function (modeled as a random oracle) to the player's view and additional randomness.
2. Using the Fiat-Shamir transform, send the $3R$ commitments and output shares of each player in all runs to a random oracle (implemented as a hash function).

3. Interpret the output of the random oracle as a challenge $\{e^{(i)}\}_{i=1}^{R}$. For each run $i \in [1, R]$, the challenge element $e^{(i)} \in \{1, 2, 3\}$ specifies to open the views of the two players $P^{(i)}_{e^{(i)}}, P^{(i)}_{e^{(i)}+1}$ (where $3 + 1 = 1$).
4. The proof contains for each run $i \in [1, R]$:
   - The commitments and output shares of all 3 players.
   - The two views and commitment openings (i.e., additional randomness) of the players $P^{(i)}_{e^{(i)}}, P^{(i)}_{e^{(i)}+1}$, indicated by the challenge.
   - The values $k^{(i)}_{e^{(i)}}, k^{(i)}_{e^{(i)}+1}$. Namely, the random seeds used by the two players whose views are opened.

Due to the 2-privacy property, opening two views for each run does not leak information about the witness. The number of runs, R, is chosen to achieve negligible soundness error, i.e., it should be infeasible for the prover to cheat without getting caught in at least one of the runs. More specifically, in order to achieve soundness error of $2^{-\kappa}$, we set $R = \lceil \kappa(\log_2 3 - 1)^{-1} \rceil$.

The verifier checks that: (1) for each run, the output shares of the three views reconstruct to y, (2) for each run, each of the two open views was computed correctly and their commitment openings are valid, and (3) the challenge was computed correctly,

In the following, we describe Step 1.(a) in the above ZKBoo protocol in more detail.

2.2 (2, 3)-Function Decomposition

ZKBoo uses the following circuit decomposition.

Definition 1. *Let f be a function that is computed by an N-gate circuit ϕ such that $f(x) = \phi(x) = y$, and let κ be the security parameter. Let k_1, k_2, k_3 be seeds chosen uniformly at random from $\{0,1\}^\kappa$, corresponding to players P_1, P_2, P_3, respectively. A $(2, 3)$-decomposition of ϕ is a tuple of algorithms $\mathcal{D} =$ (Share, Update, Output, Reconstruct):*

- $(\text{view}_1^{(0)}, \text{view}_2^{(0)}, \text{view}_3^{(0)}) \leftarrow \text{Share}(x, k_1, k_2, k_3)$
  On input of the secret value x and random seeds, outputs the initial views for each player containing the secret share x_i of x.
- $\text{view}_i^{(j+1)} \leftarrow \text{Update}(\text{view}_i^{(j)}, \text{view}_{i+1}^{(j)}, k_i, k_{i+1})$
  On input of the views $\text{view}_i^{(j)}, \text{view}_{i+1}^{(j)}$ and random seeds k_i, k_{i+1}, computes wire values for the next gate and returns the updated view $\text{view}_i^{(j+1)}$.
- $y_i \leftarrow \text{Output}(\text{view}_i^{(N)})$
  On input of the final view $\text{view}_i^{(N)}$, returns the output share y_i.
- $y \leftarrow \text{Reconstruct}(y_1, y_2, y_3)$
  On input of output shares y_i, reconstructs and returns y.

In order to compute a run for the computation $\phi(x)$ using the decomposition $\mathcal{D}$ defined above, the prover executes the steps detailed below.

1. Choose the seeds k_1, k_2, k_3 uniformly at random from $\{0,1\}^\kappa$.
2. $(\text{view}_1^{(0)}, \text{view}_2^{(0)}, \text{view}_3^{(0)}) \leftarrow \text{Share}(x, k_1, k_2, k_3)$
3. For each of the three views, call the Update function successively for every gate in the circuit:

$$\text{view}_i^{(j+1)} \leftarrow \text{Update}(\text{view}_i^{(j)}, \text{view}_{i+1}^{(j)}, k_i, k_{i+1}),$$

   for $i \in \{1,2,3\}$, $j \in [1, N]$.
4. From the final views, compute the output share of each view:

$$y_i \leftarrow \text{Output}(\text{view}_i^{(N)}),$$

   for $i \in \{1,2,3\}$.
5. $y \leftarrow \text{Reconstruct}(y_1, y_2, y_3)$

The correctness property requires that the output y above satisfies $y = \phi(x)$. The 2-privacy property requires that revealing the views of any two players reveals nothing about x.

2.3 The ZKBoo $(2,3)$-Function Decomposition

The ZKBoo protocol works over some finite ring $\mathbb{R}$. Let $f : \mathbb{R}^m \to \mathbb{R}^\ell$ be a function and ϕ an arithmetic circuit realizing f with N gates that include addition by constant, multiplication by constant, binary addition and binary multiplication gates. The $(2,3)$-decomposition of ϕ in ZKBoo is a *linear decomposition*: denote by w_k the value of the k'th wire of ϕ. Then, each party P_i has a corresponding wire value $w_k^{(i)}$. The linear decomposition maintains the invariant that for all wires, $w_k = w_k^{(1)} + w_k^{(2)} + w_k^{(3)}$. In detail, the $(2,3)$-decomposition is defined using the following tuple of algorithms:

- $\text{Share}(x, k_1, k_2, k_3)$: Samples uniform $x_1, x_2 \in \mathbb{R}^m$ and computes x_3 such that $x_1 + x_2 + x_3 = x$ (or $x_3 = x - x_1 - x_2$). Returns views containing x_1, x_2, x_3.
- $\text{Update}(\text{view}_i^{(j)}, \text{view}_{i+1}^{(j)}, k_i, k_{i+1})$: Computes P_i's view of the output wire of gate g_j and appends it to the view. For the k'th wire w_k (where $w_k^{(i)}$ denotes P_i's view for the wire), the update operation is defined as follows:

  Addition by constant: $(w_b = w_a + d)$: $w_b^{(i)} = w_a^{(i)} + d$ if $i = 1$ and $w_b^{(i)} = w_a^{(i)}$, otherwise.

  Multiplication by constant: $(w_b = w_a \cdot d)$: $w_b^{(i)} = w_a^{(i)} \cdot d$.

  Binary addition: $(w_c = w_a + w_b)$: $w_c^{(i)} = w_a^{(i)} + w_b^{(i)}$.

  Binary multiplication: $(w_c = w_a \cdot w_b)$:

$$w_c^{(i)} = (w_a^{(i)} \cdot w_b^{(i)}) + (w_a^{(i+1)} \cdot w_b^{(i)}) + (w_a^{(i)} \cdot w_b^{(i+1)}) + R_i(c) - R_{i+1}(c),$$

  where $R_i(c)$ is the c'th output of a pseudorandom generator (PRG) seeded with k_i.

– Output($\text{view}_i^{(N)}$): Returns the output wires of view, $\text{view}_i^{(N)}$.
– Reconstruct(y_1, y_2, y_3): Returns $y = y_1 + y_2 + y_3$.

It is easy to verify that the decomposition maintains the invariant $w_k = w_k^{(1)} + w_k^{(2)} + w_k^{(3)}$ for all wires, which implies that it is correct. Note that P_i can compute all gate types locally with the exception of binary multiplication gates which require inputs from P_{i+1}.

Serializing the Views It is sufficient for the prover to include in the proof only the wire values of the gates that require non-local computations (namely, the binary multiplication gates). The verifier can recompute these omitted parts of the view by local computations (i.e., they do not need to be serialized). In ZKBoo, a serialized view includes: (1) the input share, (2) output wire values for binary multiplication gates, and (3) the output share.

2.4 ZKB++

ZKB++ is an improved version of ZKBoo, obtained using several optimizations which reduce the proof size to less than a half. In general, these optimizations mainly show that some values included in the ZKBoo proof (as outlined above) can be directly computed by the verifier and hence can be omitted in the proof of ZKB++. In our context, most of these optimization are not very relevant as the attacker (verifier) has access to all data included in the original ZKBoo proof (since it is either directly included in the shorter ZKB++ proof, or can be easily computed from it).

The only optimization that is directly exploited in our attack involves the Share function: instead of uniformly sampling the input shares x_1, x_2, the Share function of ZKB++ uses pseudo-random shares for the first 2 players, generated by PRG invocations seeded with the corresponding player's random seed (k_1 or k_2). Since the random seeds of two players are revealed in the proof, the verifier can compute some of the shares (the ones of the first two players whose seeds are revealed) using the known seeds and they do not have to be included in the proof.

2.5 The Picnic Signature Scheme

The Picnic signature scheme is based on the ZKB++ protocol, where the input to the hash function that computes the challenge also includes the message m to be signed (in addition to the $3R$ commitments and output shares of each player, which are input to the hash function in ZKB++).[9]

In order to define the statement to be proved by the signer, Picnic uses a block cipher, Enc. In the classical setting (on which we focus in this paper), the block size of the block cipher and its key size in bits are both equal to the security parameter κ.

[9] In the NIST submission, the public key is hashed as well.

During key generation, the signer chooses a plaintext p and a key x for the block cipher uniformly at random from $\{0,1\}^\kappa$, encrypts the plaintext using the key and obtains the ciphertext y (of length κ bits). The public key is the plaintext-ciphertext pair (p, y) and the private signing key is the pair (x, p) (i.e., the chosen block cipher key and plaintext).

During signing, the signer proves knowledge of the key x, which encrypts p to y. Namely, Picnic uses ZKB++ in order to prove knowledge of a witness for the relation $Re := \{((p, y), x), \mathrm{Enc}_x(p) = y\}$, where $\mathrm{Enc}_x(p)$ is the block cipher encryption of plaintext p with of the key x.

The specific block cipher used by Picnic is LowMC [1], implemented using a Boolean circuit. LowMC is an iterative block cipher that employs a certain number of encryption rounds to its input. The most relevant components of LowMC for this paper are its identical 3×3 Sboxes (all the other operations are linear over $GF(2)$). Each LowMC round applies a certain number of Sboxes in parallel to the encryption state. In all LowMC variants used in Picnic, 10 parallel Sboxes are applied in a round. The algebraic normal form of an Sbox is given as

$$S(w_{a_1}, w_{a_2}, w_{a_3}) = \\ \big(w_{a_1} \oplus (w_{a_2} \cdot w_{a_3}), w_{a_1} \oplus w_{a_2} \oplus (w_{a_1} \cdot w_{a_3}), w_{a_1} \oplus w_{a_2} \oplus w_{a_3} \oplus (w_{a_1} \cdot w_{a_2})\big). \tag{1}$$

In particular, the Sbox employs 3 non-linear AND operations

$$w_{a_2} \cdot w_{a_3}, w_{a_1} \cdot w_{a_3}, w_{a_1} \cdot w_{a_2}$$

in computing the 3 output bits, respectively.

Picnic defines a total of 6 instances depending on a desired security level and on whether they are intended to resist attacks by quantum computers. We focus on the instances that are deemed secure (only) against attacks by classical computers, whose parameters are given in Table 1. However, our attacks are applicable to all instances. Note that all LowMC instances have at least 200

Table 1. Picnic Instances (for classical security)

Instance	κ	LowMC rounds	Sboxes\round	PRG	R
picnic-L1-FS	128	20	10	SHAKE128	219
picnic-L3-FS	192	30	10	SHAKE256	324
picnic-L5-FS	256	38	10	SHAKE256	438

Sboxes, where each Sbox employs 3 AND operation. Since evaluating an AND operation in Picnic requires a PRG output bit from each player, then each player computes at least $200 \cdot 3 = 600$ PRG output bits during a run.

3 The KKW Protocol [13]

In this section we give a very brief overview of the KKW protocol [13], focusing on details relevant for the paper.

The KKW protocol describes a new way to instantiate the MPC-in-the-head approach which leads to shorter proofs compared to ZKB++. The main idea is to instantiate the MPC protocol in the preprocessing model, which makes it possible to use protocols designed for a large number of players with small communication complexity (which translates to small proofs in the ZK proof protocol) and low soundness error per protocol execution (i.e., run). In the following, we only partially summarize the details of KKW's MPC protocol and refer the reader to the original paper [13] for more details about the full protocol.

The KKW MPC protocol involves n players that compute a Boolean circuit on the secret input x. The privacy property of the protocol assures that revealing the states and randomness of $n-1$ (all-but-one) players reveals nothing about the secret input x. The protocol maintains the invariant that, for each wire in the circuit α, the players hold an n-out-of-n XOR-based secret sharing of a random mask λ_α, denoted by $[\lambda_\alpha]$, along with the public masked value of the wire $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$ on the input x.

During the preprocessing phase, shares are distributed among the players as follows. For each wire α that is either an input wire of the circuit or the output wire of an AND gate, the players are given $[\lambda_\alpha]$, where $\lambda_\alpha \in \{0,1\}$ is uniform. For an XOR gate with input wires α, β and output wire γ, let $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$ (the players can compute $[\lambda_\gamma]$ locally). Finally, for each AND gate with input wires α, β, the players are given $[\lambda_{a,b}]$, where $\lambda_{\alpha,\beta} = \lambda_\alpha \cdot \lambda_\beta$.

The uniform shares of $\{\lambda_\alpha\}$ are generated by each player P_i by applying a PRG to its short input seed $k_i \in \{0,1\}^\kappa$ (where κ is the claimed security level). Then, each $\{\lambda_\alpha\}$ is defined implicitly by these shares. The shares of each $\{\lambda_{a,b}\}$ are also generated this way, but the final shares of P_n are constrained by the values of $\{\lambda_\alpha\}$. Therefore, P_n is given additional $|C|$ "correction bits" (where $|C|$ is the number of AND gates in the circuit) that determine its share of $\{\lambda_\alpha\}$ for each AND gate.

In the online phase, the players are given a masked value $\hat{z}_\alpha$ for each input wire α. The players inductively compute $\hat{z}_\alpha$ for all wires in the circuit. The full details of the online protocol are given in [13]. We remark that when used to instantiate MPC-in-the-head, an unopened player $i \in [1, n]$ is selected, while the views of the remaining $n-1$ players are opened. For each player this involves revealing its secret seed, while for P_n this additionally involves revealing the auxiliary $|C|$ correction bits.

4 Multi-Target Attacks on Zero-Knowledge Protocols

4.1 Outline of the Attacks

In this section we give a general overview of our multi-target attacks on the KKW protocol and Picnic. We assume that the attacker has access to $D = 2^d$ runs of

the underlying MPC-in-the-head protocol (generated by a single of by multiple users). In each run, the views of all-but-one player are opened, along with their randomness. We refer to the player whose view is not opened as the unopened player. The randomness used by each player is generated by a PRG initialized with a seed of length κ bits, where κ is the claimed security level against classical attack algorithms. A crucial assumption required for the multi-target attacks is that for each run, the attacker can extract a string of bits output by the PRG of the unopened player.

Below, we provide a very rough outline of the steps of the multi-user multi-target attack and its analysis in the setting described above.

1. For each run $r \in [1, 2^d]$, extract a string of bits b_r that are output by the PRG of the unopened player, and store b_r along with run r.
2. For each PRG seed $k \in [1, 2^{\kappa-d}]$,[a] derive a corresponding PRG output string b'_k using the seed k, and compare with the 2^d stored strings b_r.
3. For each matching pair r, k such that $b_r = b'_k$, compute and output the corresponding secret witness x.

[a] The seed values can be selected arbitrarily.

After trying $2^{\kappa-d}$ random seeds to Step 2, according to the birthday paradox, the attacker will test a seed used in one of the 2^d runs with high probability (assuming that the players' seeds are selected uniformly at random). Given that in Step 1 the attacker can extract sufficiently many PRG output bits from each run,[10] then the expected number of matches will be (a small) constant. Finally, assuming that Step 3 can indeed be performed, the attacker will recover the secret witness for the corresponding run. In the information theoretic model assumed in the security analysis of Picnic and KKW, the complexity of the attack is $2^{\kappa-d}$ invocations of the PRG (as long as $d \leq \kappa/2$). However, in practice the computational complexity could be higher, depending on how efficiently the matching in Step 2 is performed.

Next, we describe each one of these steps for the KKW protocol. The dedicated attack on Picnic (detailed in Section 5) uses a variant of the attack above in order to optimize its complexity. In particular, for a range of parameter values, it filters out some of the 2^d runs in the first step and keeps only those that satisfy a certain condition which allows more efficient matching in the second step.

4.2 A Multi-Target Attack on the KKW protocol

We describe the step details of the multi-target attack on the KKW protocol. In contrast to our analysis of Picnic, we will not calculate the concrete (computational) complexity of the attack. In particular, we will reduce the second step of

[10] In general, κ bits are sufficient to uniquely determine the key on average. However, even if several candidate keys are recovered, they can be filtered against the public key.

the attack to a known problem, but will not analyze the known algorithms for this problem in order to determine the best one for a given set of parameters.

Step 1: Deriving PRG Output of the Unopened Player We focus on the additive secret sharing of $\lambda_{a,b} = \lambda_a \cdot \lambda_b$. We assume that the view of P_i is unopened for $i \neq n$, hence the attacker has all shares of $\lambda_{a,b}$, except for the i'th share that is computed using a PRG applied to the seed of unopened P_i. Observe that $\lambda_{a,b} = \lambda_a \cdot \lambda_b$ is not uniform, as it is equal to 0 with probability 3/4. Consequently, the attacker can compute a guess for P_i's share of $\lambda_{a,b}$, which is correct with probability 3/4 by XORing together all the known $n-1$ shares. Hence, in this case, the attacker does not obtain direct outputs of P_i's pseudo-random bits, but rather noisy bits with a noise of 1/4.

Step 2: Matching a Run and a PRG Seed According to the previous step, finding a match between the 2^d runs and $2^{\kappa-d}$ PRG seeds reduces to finding a pair of highly correlated strings (with expected correlation of 3/4) among two groups of strings (which, other than the matching pair, are assumed to be independent and uniform). This is known as the nearest neighbor search problem. The trivial algorithm for this problem simply exhausts all string pairs and runs in time $2^{\kappa-d} \cdot 2^d = 2^\kappa$. However, there are more efficient algorithms for this well-studied problem (cf. [17, 22]).

Step 3: Recovering the Secret Witness Given a run and a seed for the unopened player, we can compute all random bits used by this player in the run.

In the KKW protocol, for each wire in the circuit, the players holds an n-out-of-n secret sharing of a random mask along with masked value of the wire, which is public (and given to the players in the online execution of the protocol). In particular, this applies to the input wires, whose value encodes the bits of the secret witness x. The randomness of the unopened player allows the attacker to compute the missing share for each wire α of x, and thus compute the random mask λ_α for this wire by summing together (XORing) all the n shares for the mask $[\lambda_\alpha]$. Finally, the attacker XORs the mask λ_α with the public masked value of the wire $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$, which gives the value of the corresponding bit of x, z_α.

5 The Multi-Target Attack on Picnic

The attack on Picnic is a variant of the general attack of Section 4.1. In this section, we describe it in more detail and start with an overview below.

5.1 Overview of the Attack

Given $D = 2^d$ runs, our goal is to devise a concrete attack on Picnic by matching the PRG output of the unopened player in each run with output obtained by evaluating the PRG with arbitrary seeds (similarly to the generic attack described in

Section 4). If each run would contain values about the same PRG output bits, we could sort these values and efficiently match each PRG evaluation with the runs. However, as we will see later each run contains data about different bits of the PRG output of the corresponding unopened player (and the number of known bits varies according to the run). Based on this fact, we describe below a more specific (yet still incomplete) outline of the steps, parameterized by κ, d, d', ℓ.

1. Out of 2^d runs, filter out ones that contain less data (about the PRG output of the unopened player) than some threshold.
2. For of each remaining run, $r \in [1, 2^{d'}]$: extract a prefix of ℓ bits that are output by the PRG of the unopened player (including possible unknown bits). Enumerate over all possible guesses for the unknown bits in the prefix, and store all the generated fully specified ℓ-bit *expanded strings* in a hash table (with a pointer to run r).
3. For each PRG seed $k \in [1, 2^{\kappa-d'}]$: derive an ℓ-bit PRG output string using the seed k, and search for it in the hash table. For each match: obtain the corresponding run r and compare the additional PRG output bits computed from this run with the PRG output. In case of equality, compute and output the corresponding secret key x.

Analysis Sketch We briefly analyze the attack for the specific case where we wish to obtain the information theoretic complexity of $2^{\kappa-d}$ (assuming $d \leq \kappa/2$). In this case, we must have $d = d'$, i.e., we cannot use any filtering in Step 1.

We introduce another parameter $0 < \tau \leq 1$, which quantifies the fraction of bits that we can extract from each run about the ℓ-bit prefix of the PRG output of the unopened player. Namely, we assume that for an ℓ-bit prefix, we can determine $\tau\ell$ bits, while $(1-\tau)\ell$ are unknown.[11] Hence, for each run, we obtain $2^{(1-\tau)\ell}$ expanded strings in Step 2 and the hash table contains a total of $2^{d+(1-\tau)\ell}$ strings of ℓ bits. We refer to τ as the *information rate* that we can achieve.

Given a random ℓ-bit PRG output in Step 3, the expected number of matches with the hash table is $2^{-\ell} \cdot 2^{d+(1-\tau)\ell} = 2^{d-\tau\ell}$, hence the total number of matches tested in the attack (before the key is recovered) is $2^{\kappa-d} \cdot 2^{d-\tau\ell} = 2^{\kappa-\tau\ell}$.

Taking into account all the steps, the expected complexity of the attack is $max(2^{\kappa-d}, 2^{d+(1-\tau)\ell}, 2^{\kappa-\tau\ell})$. We balance the first and third terms by setting $\tau\ell = d$, or $\ell = d/\tau$. Then, the complexity becomes $max(2^{\kappa-d}, 2^{d/\tau})$, which implies that information theoretic complexity can be obtained as long as $d/\tau \leq \kappa - d$, or $d \leq \kappa \cdot (\tau/(1+\tau))$. The optimal complexity in this case is $2^{\kappa-d} = 2^{\kappa(1/(1+\tau))}$. When $d > \kappa \cdot (\tau/(1+\tau))$, the information theoretic complexity cannot be reached, and we will apply filtering to optimize the complexity.

[11] We assume here for that sake of simplicity that τ is constant and does not depend on the analyzed run (although as we will see later, this does not necessarily hold).

Optimizations and Parameters for the Attack Clearly, the complexity of the attack depends in a strong way on the information rate τ, namely, on the ability to extract as much information as possible from each run about the PRG output of the unopened player. The first part of the concrete analysis below (which is the most technical one) involves deriving methods that maximize the information rate. We first show that a naive method achieves $\tau = 1/4$, giving (optimal) complexity of $2^{\kappa(1/(1+\tau))} = 2^{4\kappa/5} \approx 2^{102}$ for $\kappa = 128$. We then utilize the design of Picnic (and the underlying LowMC circuit) in order to maximize the information rate. In particular, we obtain $\tau = 1/2$, which significantly improves the complexity to $2^{2\kappa/3} \approx 2^{85}$ for $\kappa = 128$. Finally, by applying filtering, we reduce the optimal complexity to about 2^{77}.

As a concrete example of the parameters, we note that our optimized attack has $\tau \geq 1/2$, hence we need to match $\ell = d/\tau < 2d$ PRG output bits. In this paper, we only consider data complexity of $d < 64{+}9 = 73$, hence $\ell < 2{\cdot}73 = 146$. We note that these PRG output bits are used in the evaluation of $\lceil 146/3 \rceil = 49$ Sboxes, whereas all LowMC variants in Picnic have at least 200 Sboxes.

5.2 Deriving PRG Output of the Unopened Player

We start by describing a preliminary method to extract PRG output of the unopened player. We then present two optimized methods, exploiting the specific Sbox design of LowMC. In Appendix A we describe an additional method which is not directly used in our attack, but is interesting nevertheless.

Preliminary Extraction Method We consider a binary multiplication gate ($w_c = w_a \cdot w_b$). Recall that in ZKBoo (and ZKB++),

$$w_c^{(i)} = (w_a^{(i)} \cdot w_b^{(i)}) + (w_a^{(i+1)} \cdot w_b^{(i)}) + (w_a^{(i)} \cdot w_b^{(i+1)}) + R_i(c) - R_{i+1}(c). \quad (2)$$

Let us assume that the views and random seeds of players $2, 3$ are revealed. Consider $i = 3$, for which the equation above reduces to:

$$w_c^{(3)} = (w_a^{(3)} \cdot w_b^{(3)}) + (w_a^{(1)} \cdot w_b^{(3)}) + (w_a^{(3)} \cdot w_b^{(1)}) + R_3(c) - R_1(c).$$

Moreover, we assume that

$$w_a^{(3)} = w_b^{(3)} = 0. \quad (3)$$

Note that since view 3 is revealed, the attacker knows when this event occurs. Conditioned on this event, the equation simplifies to $w_c^{(3)} = R_3(c) - R_1(c)$, or

$$R_1(c) = R_3(c) - w_c^{(3)}.$$

Since $R_3(c)$ and $w_c^{(3)}$ are known from random seed and view of player 3 (respectively), then the attacker can compute $R_1(c)$ with probability 1, conditioned on (3).

In Boolean circuits (as in Picnic), we expect (generalized) condition (3) to hold for 1/4 of the AND gates (the probability is over the randomness of the view of P_{e+1}). Consequently, the attacker knows about $\tau = 1/4$ of the output bits produced by the PRG (not including the ones used in the initial Share function). Note that the locations of these known output bits depends on $w_a^{(e+1)}$ and $w_b^{(e+1)}$, which are different for each run.

Below, we exploit the specific structure of the Picnic circuit in order to optimize the information rate.

Extraction Method 1 Recall that the AND operations performed by a LowMC Sbox are

$$S'(w_{a_1}, w_{a_2}, w_{a_3}) = (w_{a_2} \cdot w_{a_3}, w_{a_1} \cdot w_{a_3}, w_{a_1} \cdot w_{a_2}),$$

where S' denotes the function obtained from S by only considering AND operations. Denote the output wires of these 3 AND operations by $w_{c_1}, w_{c_2}, w_{c_3}$, respectively, and write the basic equation of (2) with $i = 3$ for the 3 AND gates:

$$\begin{aligned} w_{c_1}^{(3)} &= (w_{a_2}^{(3)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_2}^{(1)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_2}^{(3)} \cdot w_{a_3}^{(1)}) \oplus R_3(c_1) \oplus R_1(c_1), \\ w_{c_2}^{(3)} &= (w_{a_1}^{(3)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_1}^{(1)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_1}^{(3)} \cdot w_{a_3}^{(1)}) \oplus R_3(c_2) \oplus R_1(c_2), \\ w_{c_3}^{(3)} &= (w_{a_1}^{(3)} \cdot w_{a_2}^{(3)}) \oplus (w_{a_1}^{(1)} \cdot w_{a_2}^{(3)}) \oplus (w_{a_1}^{(3)} \cdot w_{a_2}^{(1)}) \oplus R_3(c_3) \oplus R_1(c_3). \end{aligned} \quad (4)$$

Assuming the the views of players $2, 3$ are revealed, the unknown values in these 3 equations are the view and randomness variables of player 1:

$$R_1(c_1), R_1(c_2), R_1(c_3), w_{a_1}^{(1)}, w_{a_2}^{(1)}, w_{a_3}^{(1)}.$$

Observe that for every value of the 3 known bits $w_{a_1}^{(3)}, w_{a_2}^{(3)}, w_{a_3}^{(3)}$, we obtain a linear equation system with 3 equations. Our goal is to perform Gaussian elimination on this system in order to eliminate the unknown variables $w_{a_1}^{(1)}, w_{a_2}^{(1)}, w_{a_3}^{(1)}$ and remain with linear relations in the 3 randomness variables of player 1, $R_1(c_1), R_1(c_2), R_1(c_3)$. We are thus interested in the rank of the equation system in $w_{a_1}^{(1)}, w_{a_2}^{(1)}, w_{a_3}^{(1)}$ as a function of the known variables $w_{a_1}^{(3)}, w_{a_2}^{(3)}, w_{a_3}^{(3)}$.

Since the equation system is symmetric, the rank depends only on the Hamming weight (HW) of $w_{a_1}^{(3)}, w_{a_2}^{(3)}, w_{a_3}^{(3)}$. It is easy to check that the following holds:

- If $HW = 0$ (i.e., $w_{a_1}^{(3)} = w_{a_2}^{(3)} = w_{a_3}^{(3)} = 0$), the rank is 0 and we obtain the 3 PRG output bits $R_1(c_1), R_1(c_2), R_1(c_3)$.
- If $HW > 0$, the rank is 2 and we obtain 1 PRG output bit (or a linear combination of output bits) according to the specific values of $w_{a_1}^{(3)}, w_{a_2}^{(3)}, w_{a_3}^{(3)}$.

The first case $w_{a_1}^{(3)} = w_{a_2}^{(3)} = w_{a_3}^{(3)} = 0$ occurs with probability 1/8 (over the randomness of the view of P_3). Note that the equation system is never of full rank and we can always obtain at least 1 bit of information about $R_1(c_1), R_1(c_2), R_1(c_3)$.

Example 1. If $w_{a_1}^{(3)} = w_{a_2}^{(3)} = w_{a_3}^{(3)} = 1$, we XOR together the 3 equations to eliminate $w_{a_1}^{(1)}, w_{a_2}^{(1)}, w_{a_3}^{(1)}$, and can compute the value of $R_1(c_1) \oplus R_1(c_2) \oplus R_1(c_3)$.

We performed the analysis assuming the views of P_2, P_3 were opened in the run, but similar analysis applies (with appropriate indexing modifications) regardless of which 2 views are opened. We summarize our findings below.

Proposition 1. *Given access to the open views and randomness of P_e, P_{e+1} in Picnic, for any triplet of wires that are input to a LowMC Sbox and its corresponding triplet of output wires $w_{c_1}, w_{c_2}, w_{c_3}$, with probability 1/8 (over the randomness of the view of P_{e+1}), we can easily compute the corresponding PRG output bits of P_{e+2}, namely, $R_{e+2}(c_1), R_{e+2}(c_2), R_{e+2}(c_3)$. Otherwise (with probability 7/8) we can compute one of seven possible linear equations on these bits, where each particular linear equation is obtained with probability 1/8 (over the randomness of the view of P_{e+1}).*

Hence, with high probability, for *most runs* we obtain 3 bits of information for at least 1/8 of the Sboxes and 1 bit of information for the remaining Sboxes. We therefore obtain at least $3/8 + 7 \cdot 1/8 = 10/8 = 5/4$ bits of information on average per 3-bit Sbox, or $\tau_1 = 5/12$ bits of information per PRG output bit (for most runs). This is significantly better than the ratio of 1/4 obtained in a generic manner above.

Extraction Method 2 Assume that P_2, P_3 are opened and reconsider the equation system of (4). If we guess the values of $w_{a_1}, w_{a_2}, w_{a_3}$, we can easily deduce the unknown PRG output bits $R_1(c_1), R_1(c_2), R_1(c_3)$ by first computing $w_{a_1}^{(1)}, w_{a_2}^{(1)}, w_{a_3}^{(1)}$. On its own, this is a useless observation since we could have directly guessed these 3 PRG output bits. However, let us assume that the analyzed Sbox is located in the first LowMC Sbox layer. This implies that each of $w_{a_1}, w_{a_2}, w_{a_3}$ is a linear function of the unknown LowMC secret key x corresponding to the run (as there is no non-linear function applied to compute these bits from the known plaintext). Therefore, the knowledge of the 3 bits $w_{a_1}, w_{a_2}, w_{a_3}$ input to the Sbox directly translates to knowledge of 3 linear equations on the LowMC secret key. More specifically, we have $w_{a_i} = l_{a_i}(x)$ for $i \in \{1, 2, 3\}$, where $l_{a_i}(x)$ is a linear equation on the secret key x. Recall that the Share function outputs 3 shares that sum to the LowMC secret key $x = x_1 \oplus x_2 \oplus x_3$. Therefore, for $i \in \{1, 2, 3\}$ we have

$$w_{a_i} = l_{a_i}(x) = l_{a_i}(x_1) \oplus l_{a_i}(x_2) \oplus l_{a_i}(x_3), \text{ or}$$

$$l_{a_i}(x_1) = w_{a_i} \oplus l_{a_i}(x_2) \oplus l_{a_i}(x_3). \tag{5}$$

We (assume to) know $w_{a_i}, l_{a_i}(x_2), l_{a_i}(x_3)$, and therefore, we can derive $l_{a_i}(x_1)$. The 3 bits $l_{a_i}(x_1)$ are linear combinations of PRG output bits of P_1 that are output by the Share function, and we have shown that they are directly deduced from the knowledge of w_{a_i}.

Altogether, we guess 3 bits and obtain 6 PRG output bits. Crucially for the attack, that indices of the computed 6 PRG (linear combinations of) output bits are fixed among all runs.

Proposition 2. *Given access to the open views and randomness of P_e, P_{e+1} in Picnic for $e \in \{2,3\}$, for any triplet of wires that are input to a LowMC Sbox in the first Sbox layer, a guess for the 3 bit values for these wires allows to easily compute a guess for values of 6 (linear combinations of) PRG output bits of P_{e+2}. The locations of these output bits depend only on the choice of Sbox. Moreover, the same holds for any Sbox in the i'th Sbox layer, given that we have a guess for the all the Sbox inputs in layers $1, 2, \ldots, i-1$.*

Note that the proposition only applies to $e \in \{2,3\}$, as for $e = 1$, the key share of $P_{e+2} = P_3$ (namely x_3) is not computed using a PRG. The last part of the proposition holds since each input wire to each Sbox in the i'th layer can be expressed as a linear combination of the key bits and the output bits of the Sboxes in previous layers. These output bits are (assumed to be) known.

Recall that in the previous extraction method we obtained an information rate of $\tau_1 = 5/12$ for most runs. Here, we guess 3 bits and obtain 6 PRG output bits, i.e. $\tau_2 = 1/2 > \tau_1$, and hence this method can be viewed as an improvement over the previous one. On the other hand, for specific runs which deviate from the average, the first method may yield a higher information rate, thus the methods are not always directly comparable. Indeed, our attack will combine these methods according to some parameter values.

5.3 Exploiting PRG Output of the Unopened Player

We focus on a single run that contains data about the PRG output of the unopened player (P_{e+2}) . We only exploit data for runs with $e \in \{2,3\}$ in the attack.

We call the useful data extracted from a run a *target string* (TS). The target string is indexed according to triplets of bits (corresponding to the Sboxes in LowMC's circuit), where the relevant information about each triplet consists of the known view and randomness bits of P_{e+1}. For example, if $e = 2$ (i.e., players 2,3 are opened), then for each Sbox in LowMC's circuit, the TS contains all the known view and randomness bits of P_3 that appear in the equation system of (4). Recall from Proposition 1 that for each triplet, we may obtain all the 3 PRG output bits of the unopened player, and in this case, we call it a *full triplet*. Otherwise, we obtain 1 bit of a linear equation on the 3 PRG output bits (the linear equation itself depends on the view of P_{e+1}) and call the triplet a *partial triplet*.

The triplet data in each target string is sorted according to the indices of LowMC's Sboxes. Importantly, Sboxes in each layer i appear together before layer $i+1$ (the order within each layer is chosen arbitrarily, but is consistent among all target strings). Given a target string ts, we refer to the data of the i'th triplet by $ts[i]$, and to the data of the triplet sequence $i, i+1, \ldots, j$ as $ts[i,j]$.

For the purpose of exploiting Proposition 2, we also need auxiliary information from the Share function about the shares of P_e, P_{e+1}. We append this data to each TS.

Target String Expansion We elaborate on Step 2 of the general attack of Section 5.1 by defining the expansion of a target string ts. Essentially, it is a set of strings that correspond to all possible PRG outputs of the unopened player (for some specific triplets) that match the partial information in ts.

Given parameters $t_1, t_2 \geq 0$, assume $ts[t_1+1, t_1+t_2]$ contains $t_3 \leq t_2$ full triplets (and $t_2 - t_3$ partial triplets). We can expand the t_2 triplets of $ts[t_1 + 1, t_1 + t_2]$ according to Proposition 1 into a set of $2^{2(t_2-t_3)}$ expanded strings, each of length $3t_2$ bits. Similarly, we can expand triplets $ts[1, t_1]$ according to Proposition 2 into a set of 2^{3t_1} expanded strings, each of length $6t_1$ bits.

Combining these two expansion methods into one expansion function (denoted $expand(ts)$), we obtain a set of $2^{3t_1+2(t_2-t_3)}$ strings. Each string is of length $\ell = 6t_1 + 3t_2$ bits and represents possible values for certain $6t_1 + 3t_2$ PRG output bits, which we call *matching bits* (mb). Given a PRG seed k, we denote by $PRG_k[mb]$ the $6t_1 + 3t_2$-bit PRG output value for the matching bits, when evaluated with k. Given a TS, ts, generated with seed k, only one of the $2^{3t_1+2(t_2-t_3)}$ strings in $expand(ts)$ is equal to $PRG_k[mb]$. We summarize below.

Proposition 3. *Given a target string ts, parameters $t_1, t_2 \geq 0$, and assuming $ts[t_1 + 1, t_1 + t_2]$ contains $t_3 \leq t_2$ full triplets, the expansion of ts with parameters t_1, t_2, t_3 is a set denoted $expand(ts)$ that contains $2^{3t_1+2(t_2-t_3)}$ expanded strings, each of length $6t_1+3t_2$ bits. Each string in $expand(ts)$ contains $6t_1+3t_2$ possible values for the matching bits, derived from ts by a guess for the missing information.*

5.4 The Multi-Target Attack

It is obvious that a run with a large number of full triplets in $ts[t_1+1, t_1+t_2]$ is more useful for our purpose, as it contains more data about the PRG output of the unopened player (equivalently, its expanded set according to Proposition 3 is relatively small). Hence, we filter the target data strings, keeping those with a large value of t_3. Each remaining target string is then expanded and the resultant expanded strings are stored to be matched with data obtained from PRG evaluations. Based on Proposition 1, we derive the following proposition.

Proposition 4. *Given integer parameters $t_1 \geq 0$ and $0 \leq t_3 \leq t_2$, the probability (over the randomness of the view of P_{e+1}) that for an arbitrary target string ts, the t_2 triplets of $ts[t_1 + 1, t_1 + t_2]$ contain at least t_3 full triplets is*

$$\Gamma(t_2, t_3) \stackrel{\text{def}}{=\!=} \sum_{i=t_3}^{t_2} \binom{t_2}{t_3} \left(\frac{1}{8}\right)^i \cdot \left(\frac{7}{8}\right)^{t_2-i}.$$

We describe the attack below, using positive integer parameters $\kappa, r, r', t_1, t_2, t_3$.

1. For each of the 2^d available runs: denote by ts the target string of the current run. If $e+2=3$ (P_{e+2} is the unopened player), or if $ts[t_1+1, t_1+t_2]$ contains less than t_3 full triplets, then discard the run.
2. For of each remaining $2^{d'}$ runs: compute $expand(ts)$, and store each of the $6t_1+3t_2$-bit expanded strings $s \in expand(ts)$ (along with a pointer to ts) in a hash table L, indexed by s.
3. For each PRG seed $k \in [1, 2^{\kappa-d'}]$:
   - Evaluate the PRG, derive $PRG_k[mb]$ and search for a match in L.
   - For each match with an expanded string s, recover the corresponding ts.
   - Continue to compute the PRG output on k and compare with ts.
   - If the PRG outputs match, we have guessed the correct seed k for the unopened player in ts with high probability. Derive and output the signing key x based on the Share function $x = x_1 \oplus x_2 \oplus x_3$ by computing the missing share using k.

Analysis In order to analyze the attack, observe that on average, in 2/3 of the runs P_1 or P_2 are unopened. Out of the remaining runs, a fraction of $\Gamma(t_2, t_3)$ contains at least t_3 full triplets in $ts[t_1+1, t_1+t_2]$ (according to Proposition 4). Consequently, we expect

$$2^{d'} = 2/3 \cdot \Gamma(t_2, t_3) \cdot 2^d. \tag{6}$$

Next, according to Proposition 3, L is expected to contain at most $2^{d'+3t_1+2(t_2-t_3)}$ expanded strings, which gives the memory complexity of the attack (and a lower bound on its time complexity).

Finally, the expected number of matches in Step 3 between a random $6t_1+3t_2$-bit string $PRG_k[mb]$ and one of the expanded $2^{d'+3t_1+2(t_2-t_3)}$ strings in L is at most $2^{d'+3t_1+2(t_2-t_3)} \cdot 2^{-6t_1-3t_2} = 2^{d'-3t_1-t_2-2t_3}$. Hence, the total expected number of matches that we need to test in Step 3 is upper bounded by $2^{\kappa-d'} \cdot 2^{d'-3t_1-t_2-2t_3} = 2^{\kappa-3t_1-t_2-2t_3}$.

Taking all steps into account, the total time complexity is upper bounded by $max(2^d, 2^{\kappa-d'}, 2^{d'+3t_1+2(t_2-t_3)}, 2^{\kappa-3t_1-t_2-2t_3})$. Plugging in the value of d' calculated in (6), we obtain

$$max(2^d, 3/2 \cdot 1/\Gamma(t_2, t_3) \cdot 2^{\kappa-d}, 2/3 \cdot \Gamma(t_2, t_3) \cdot 2^{d+3t_1+2(t_2-t_3)}, 2^{\kappa-3t_1-t_2-2t_3}). \tag{7}$$

We balance the second and fourth terms by setting

$$2^{3t_1+t_2+2t_3} = 2/3 \cdot \Gamma(t_2, t_3) \cdot 2^d, \tag{8}$$

i.e., $2/3 \cdot \Gamma(t_2, t_3) = 2^{3t_1+t_2+2t_3-d}$. Thus, the third term (which also represents the memory complexity) becomes $2^{6t_1+3t_2}$ and the time complexity upper bound is calculated as

$$max(2^d, 3/2 \cdot 1/\Gamma(t_2, t_3) \cdot 2^{\kappa-d}, 2^{6t_1+3t_2}), \tag{9}$$

under constraint (8). We now analyze the complexity of the attack for various choice of the free parameters t_1, t_2, t_3.

Achieving the Information Theoretic Complexity If we want a time complexity close to the information theoretic complexity we apply a minimal amount of filtering. Based on the conclusion of Section 5.1, it is best to use the extraction method with has highest information rate. Thus, we use the second extraction method (summarized in Proposition 2), which has $\tau_2 = 1/2$, by setting $t_2 = t_3 = 0$ (and $\Gamma(t_2, t_3) = 1$). The analysis becomes similar to the one of Section 5.1, with the exception of the filtering constant $2/3$ and rounding factors. We can come close to the information theoretic time complexity and obtain time complexity of $3/2 \cdot 2^{\kappa-d}$ as long as $3/2 \cdot 2^{\kappa-d} \geq 4/9 \cdot 2^{2d}$, or

$$d \leq \log 3/2 + \kappa/3$$

(the formula only holds for values of d that satisfy $2^d = 3/2 \cdot 2^{3t_1}$ for an integer t_1).

For example, if $\kappa = 128$, we can only exploit up to $2^d \leq 3/2 \cdot 2^{42}$ data (by setting $t_1 = 14$). The optimal time complexity is therefore about $2^{128-42} = 2^{86}$. For $\kappa = 192$, we can reach the information theoretic time complexity for almost the entire range of $D \leq 324 \cdot 2^{64}$, whereas for $\kappa = 256$, we obtain the information theoretic time complexity for the full range $D \leq 438 \cdot 2^{64}$.

General Time Complexity Optimization When more than $3/2 \cdot 2^{\kappa/3}$ runs are available and our goal is to optimize time complexity, we could not accurately optimize the attack analytically as a function of the known parameters κ, d. Instead, we optimized the most precise original attack complexity equation (7) for several choices of κ, d by brute force. In particular, for $\kappa = 128$, if we restrict ourselves to 2^{64} signatures ($r = \log(219 \cdot 2^{64}) \approx 71$), then we (approximately) obtain $T = 2^{77}$ and $M = 2^{76}$ memory by setting $t_1 = 0, t_2 = 25, t_3 = 13$. This demonstrates the fact that when a large amount of data is available, we do not exploit the second extraction method in the optimal attack, namely, we set $t_1 = 0$. This is due to the fact that filtering a large amount of data results in a better information rate using the first extraction method.

We can also obtain some time-memory tradeoffs by applying more filtering. For example, we obtain $T = 2^{83}, M = 2^{57}$ by setting $t_1 = 0, t_2 = 19, t_3 = 13$.

6 Seed Collision Attack on Picnic

In this section we describe our seed collision attack on Picnic. Unlike our main multi-target attack, this attack (almost) reaches the information theoretic complexity of $2^\kappa = T \cdot D$ for $D = 2^{\kappa/2}$ (but only in the single-user setting for the particular point of $D = 2^{\kappa/2}$). The reason we can reach the information theoretic complexity here is that the matching step is much easier compared to that of our main multi-target attack.

Assume the attacker has access to two runs generated with the same private key. For run $i \in \{1, 2\}$, denote by $P^{(i)}_{e^{(i)}+2}$ the unopened player and assume that in both runs the unopened player uses the same seed $k^{(1)}_{e^{(1)}+2} = k^{(2)}_{e^{(2)}+2}$. Moreover, assume that $e^{(i)} \in \{2, 3\}$. Then, in both runs, the pseudo-random bits

generated by the PRG of the unopened players are identical, and in particular, the outputs of their Share functions are identical, namely $x^{(1)}_{e^{(1)}+2} = x^{(2)}_{e^{(2)}+2}$. Hence, $x \oplus x^{(1)}_{e^{(1)}} \oplus x^{(1)}_{e^{(1)}+1} = x \oplus x^{(2)}_{e^{(2)}} \oplus x^{(2)}_{e^{(2)}+1}$ (since both runs are generated with the same private key x). Therefore, the attacker can easily detect this event by verifying the condition

$$x^{(1)}_{e^{(1)}} \oplus x^{(1)}_{e^{(1)}+1} = x^{(2)}_{e^{(2)}} \oplus x^{(2)}_{e^{(2)}+1}. \tag{10}$$

The key recovery process (once again) has to use the available data about the PRG outputs of the unopened players. Assuming that $k^{(1)}_{e^{(1)}+2} = k^{(2)}_{e^{(2)}+2}$, the attacker can recover the secret key x by exploiting the fact that the remaining PRG outputs of $P^{(1)}_{e^{(1)}+2}$ and $P^{(2)}_{e^{(2)}+2}$ are identical, assuming that the seeds of $P^{(1)}_{e^{(1)}+1}$ and $P^{(2)}_{e^{(2)}+1}$ are different (i.e., $k^{(1)}_{e^{(1)}+1} \neq k^{(2)}_{e^{(2)}+1}$), which occurs with high probability. This is done by independently analyzing each Sbox, observing the corresponding input triplets of bits in the two target strings for the runs. Assuming that the 3-bit randomness values of $P^{(1)}_{e^{(1)}+1}$ and $P^{(2)}_{e^{(2)}+1}$ are different for the Sbox (which occurs with probability 7/8), then the attacker can always obtain two linear equations on the secret inputs to the Sbox.

Example 2. Examine again the equation system (4), assuming for simplicity that player 1 is unopened in both runs, namely, $e^{(1)} + 2 = e^{(2)} + 2 = 1$. Moreover, assume that in the first run, the 3 relevant view bits of player 3 are equal to zero, i.e., $w^{(3)}_{a_1} = w^{(3)}_{a_2} = w^{(3)}_{a_3} = 0$. Then, we can compute the common random bits of unopened player 1, namely, $R_1(c_1), R_1(c_2), R_1(c_3)$. Furthermore, assume that for the second run, the 3 relevant view bits of player 3 satisfy $u^{(3)}_{a_1} = u^{(3)}_{a_2} = 0, u^{(3)}_{a_3} = 1$ (we index these bits with u, as they are different across the runs). Then, since the randomness of player 1 is known, we can deduce the share values $u^{(1)}_{a_1}, u^{(1)}_{a_2}$ in the second run, which reveal the wire values w_{a_1}, w_{a_2}.

Linearizing the Circuit After analyzing all the Sboxes, the attacker knows 2 out of 3 (linear combinations of) input bits to a fraction of about 7/8 of the Sboxes. This makes the 3 output bits of each such Sbox linear functions of the inputs. We call these Sboxes *linearized Sboxes*. From the viewpoint of the attacker, the only non-linear operations that remain in the circuit involve the non-linearized Sboxes (whose expected fraction is 1/8). Based on this observation, we set up a linear equation system where the variables are the values of the κ key bits in addition to the 3 unknown output values of each non-linearized Sbox. Note that the value of every wire in the circuit can be expressed as a linear combination of these variables. Assuming that the circuit has K Sboxes, the expected number of variables is $\kappa + 3 \cdot K/8$.

In order to get the values of linear equations in the variables, we deduce values of specific wires (or linear combination of wires) in the circuit. First, note that the output of the circuit is known and gives rise to κ linear equations. We obtain additional equations based on the 2 known input bits of linearized

Sboxes. Hence, the expected number of equations is $\kappa + 2 \cdot 7K/8 = \kappa + 14K/8$. For Picnic, we expect to obtain many more equations than variables to the system, whose solution gives the secret signing key. For example, if $\kappa = 128$, then $K = 200$, implying that the expected number of variables is 203 and the number of equations is 478. A simple Chernoff bound shows that the numbers of variables and equations are close to their mean with high probability. For example, for $\kappa = 128, K = 200$, with probability (more than) 0.9999 the attacker knows 2 out of 3 input bits for at least $7/8 \cdot 200 - 25 = 150$ Sboxes. This implies that with probability 0.9999, the number of variables is at most $\kappa + 3 \cdot (200/8 + 25) = 278$ and the number of equations is at least $\kappa + 2 \cdot 150 = 428$.

In case the attacker is unlucky and still has to spend considerable effort in key recovery, it is possible to exploit several collisions for this purpose at the price of increased data complexity. As in typical collision attacks, the expected number of collisions grows quadratically as a function of the data. In particular, after obtaining 4 collisions (requiring about twice the amount of data), the expected fraction of Sboxes with 3 unknown input bits is $(1/8)^4 = 1/4096$. Since the LowMC instances only have a few hundreds of Sboxes, the system becomes completely linear in the key bits and the attacker directly solves for the key.

Details of the Attack The seed collision attack is described below.

1. Store each of the $\approx 2/3 \cdot 2^d$ available runs (generated with the same private key) for which $e^{(i)} \in \{2, 3\}$ in a hash table L, with the value $x^{(i)}_{e^{(i)}} \oplus x^{(i)}_{e^{(i)}+1}$ as the index.
2. For each collision $x^{(i)}_{e^{(i)}} \oplus x^{(i)}_{e^{(i)}+1} = x^{(j)}_{e^{(j)}} \oplus x^{(j)}_{e^{(j)}+1}$ between runs i, j in L:
   - If $k^{(i)}_{e^{(i)}+1} = k^{(j)}_{e^{(j)}+1}$, discard the collision.
   - Otherwise, if the known PRG output bits of runs i, j do not match, discard the collision.
   - Otherwise, recover and output the secret key by solving a system of linear equations.

Analysis Since the seeds are of a length κ bits, we need about $d = 1 + \kappa/2$ to have two runs i, j for which $k^{(i)}_{e^{(i)}+2} = k^{(j)}_{e^{(j)}+2}$ with probability larger than $1/2$ (for these runs $k^{(i)}_{e^{(i)}+1} \neq k^{(j)}_{e^{(j)}+1}$ with probability $1 - 2^{-\kappa} \approx 1$). On the other hand, $x^{(i)}_{e^{(i)}} \oplus x^{(i)}_{e^{(i)}+1} = x^{(j)}_{e^{(j)}} \oplus x^{(j)}_{e^{(j)}+1}$ is a κ-bit condition that occurs for an arbitrary pair of runs with probability $2^{-\kappa}$. Hence, summing over all pairs of runs, the expected total number of collisions that we need to test in Step 2 is about $1/4 \cdot 2^{2d-\kappa} = 1$ and the complexity of the attack is $2^d = 2 \cdot 2^{\kappa/2}$. As the linear system of equations considered in the final step has several hundreds of variables, the complexity of solving it can be bounded by 2^{30} bit operations using Gaussian elimination, and this complexity can be neglected.

The Multiple User Setting In the multiple user setting we independently run the attack on the data of each user. Assume that we have 2^u users with 2^d distinct runs available per user. Then, the success probability per user (assuming $d \leq 1 + \kappa/2$) is about $1/4 \cdot 2^{2d-\kappa}$, and the success probability across all users is roughly $1/4 \cdot 2^{u+2d-\kappa}$, implying that we need $d \approx 1 + (\kappa - u)/2$ in order to recover the key of one of the users with high (constant) probability. Therefore, we require a total of $2 \cdot 2^{u+d} = 2 \cdot 2^{(\kappa+u)/2}$ runs. More generally, if the number of available runs varies among the different users, i.e., we have 2^{d_i} available runs for user i, then the success probability is proportional to

$$1/4 \cdot 2^{-\kappa} \cdot \sum_{i=1}^{2^u} 2^{2d_i}.$$

The expression is minimized when all d_i's are equal, implying that a skewed distribution of data helps the attacker.

7 Multi-Target Attacks on Additional Cryptosystems

In this section we give two examples of optimizations used in MPC protocols that weaken their resistance to multi-target attacks. We further give an example of a general public key scheme that is vulnerable to a multi-user single-target attack. For each example, we reference at least one vulnerable scheme that was proposed recently. We note that all of the described attacks can be prevented by appropriate use of salting. This results in some performance overhead, which depends on the underlying scheme.

We assume throughout this section that the desired security level is κ bits.

7.1 Hash-Based Commitments Optimization

We consider the widely used hash-based commitment scheme, which utilizes a hash function $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$. In order to commit to a string W, one selects a sufficiently long random string $rand$, and the commitment is defined as $H(rand, W)$. The commitment is opened by revealing $rand, W$. Several MPC protocols such as [13, 16] optimize this scheme by omitting $rand$ and defining the commitment as $H(W)$, given that W has sufficient min-entropy of at least κ bits. This way, only W has to be sent when opening the commitment, thus saving communication. However, the optimization clearly exposes the protocol to multi-user multi-target attacks, as the attacker may try to derive a preimage to one out of many available commitments.

7.2 Seed Tree Optimization

We analyze an MPC protocol optimization that is used in the KKW protocol [13]. In the unoptimized protocol, the seeds of all n players are (essentially) independent and opening $n-1$ players out of n requires $\kappa \cdot (n-1)$ bits of communication.

As shown in Section 4.2, the unoptimized protocol is already vulnerable to multi-user multi-target attacks. We now consider an optimized version of the protocol described in [13], which further weakens its security against such attacks.

The optimization involves building a seed tree construction (cf. [18]) which generates seeds for the n players that participate in the MPC protocol in a way that reduces the communication required to reveal $n-1$ seeds. The seed tree is a binary tree, where each node has a label of κ bits. The label of the root is a randomly generated master seed of κ bits, and the two κ-bit labels of the 2 children of each node are defined recursively by running a PRG on the label of the parent and outputting 2κ bits. The tree is of depth $\log n$ and the seeds of the n players are defined as the labels of the n leaves. In order to reveal the seeds of all players but player i, it is sufficient to reveal the labels of the siblings of the path from the root to leaf i, which requires only $\kappa \cdot \log n$ communication.

Observe that in the original protocol, the attacker had only a single target per run, which was the seed of the unopened player. In contrast, in the optimized protocol, each node on the path from the root to the unopened leaf i is a target, as the attacker knows one of its κ-bit outputs from the $\log n$ revealed labels. Hitting one of these targets allows the attacker to easily compute the label of leaf i. The degradation in security is proportional to $\log n$, which is not large, but should still be noted.

7.3 Public Key Scheme Construction

Finally, we consider a public key scheme that uses a secret signing key $x \in \{0,1\}^\kappa$ and generates its public key as $pk = g(x)$, where g is a deterministic function. Typically, g involved invoking a PRF at least once on input x and additional (deterministically generated) inputs. An example of such a scheme is the recently proposed TACHYON signature algorithm [2] (which was presented at ACM CCS 2018). This described scheme is clearly vulnerable to a multi-user single-target attack, where the attacker obtains access to several public keys that belong to several users. The attacker attempts to recover the secret key of one (or several) of the users by iteratively guessing a value for x', computing $pk' = g(x')$, and comparing with the available public keys.

8 Conclusions

In this paper we described multi-target attacks on the Picnic signature scheme and on the related KKW protocol. Our attacks have two features that stem from Picnic's novel design and distinguish them from standard multi-target attacks:

1. The vulnerability of the cryptosystem (Picnic) to multi-target attacks is not evident, even when one carefully looks for it. As a result, it was missed by the designers.
2. Internally, the multi-target attacks cannot apply a typical sort-and-match algorithm, and efficient key recovery requires cryptanalytic effort.

The attacks expose a design weakness in the way Picnic uses a PRG during signing. Although our attacks are generally impractical, in some cases this design weakness could be leveraged in combination with an additional implementation weakness (such as generation of seeds with low entropy[12]) to mount a practical attack. Such an attack would have been harder to carry out had the PRG been appropriately salted.

Besides the short-term impact of our analysis on enhancing Picnic's security, we hope that it will contribute to the secure design of novel cryptosystems in the future.

Acknowledgements: The authors would like to thank the Picnic designers for helpful discussions about this work and anonymous reviewers for valuable suggestions.

References

1. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In Oswald and Fischlin [20], pages 430–454.
2. R. Behnia, M. O. Ozmen, A. A. Yavuz, and M. Rosulek. TACHYON: Fast Signatures from Compact Knapsack. In Lie et al. [15], pages 1855–1867.
3. D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In Oswald and Fischlin [20], pages 368–397.
4. D. Boneh, S. Eskandarian, and B. Fisch. Post-Quantum Group Signatures from Symmetric Primitives. *IACR Cryptology ePrint Archive*, 2018:261, 2018.
5. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Picnic: A Family of Post-Quantum Secure Digital Signature Algorithms. https://microsoft.github.io/Picnic/.
6. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017.
7. S. Checkoway, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, H. Shacham, and M. Fredrikson. On the Practical Exploitability of Dual EC in TLS Implementations. In K. Fu and J. Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335. USENIX Association, 2014.
8. D. Derler, S. Ramacher, and D. Slamanig. Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications to Ring Signatures from Symmetric-Key Primitives. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 419–440. Springer, 2018.

[12] The Picnic specification recommends to generate the seeds based on the (high-entropy) private key, but does not (and cannot) enforce this.

9. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.
10. I. Giacomelli, J. Madsen, and C. Orlandi. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1069–1083. USENIX Association, 2016.
11. A. Hülsing, J. Rijneveld, and F. Song. Mitigating Multi-target Attacks in Hash-Based Signatures. In C. Cheng, K. Chung, G. Persiano, and B. Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 387–416. Springer, 2016.
12. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In D. S. Johnson and U. Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30. ACM, 2007.
13. J. Katz, V. Kolesnikov, and X. Wang. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In Lie et al. [15], pages 525–537.
14. L. Lamport. Constructing Digital Signatures from a One Way Function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.
15. D. Lie, M. Mannan, M. Backes, and X. Wang, editors. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018.
16. Y. Lindell and B. Riva. Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries. In I. Ray, N. Li, and C. Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 579–590. ACM, 2015.
17. A. May and I. Ozerov. On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes. In Oswald and Fischlin [20], pages 203–228.
18. D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.
19. NIST's Post-Quantum Cryptography Project. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.
20. E. Oswald and M. Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.
21. D. Unruh. Quantum Proofs of Knowledge. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2012.
22. G. Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and the Closest Pair Problem. *J. ACM*, 62(2):13:1–13:45, 2015.

A An Additional PRG Output Extraction Method

In continuation of Section 5.2, we describe an additional (third) extraction method, which is closely related to the second one. We observe that for each Sbox in the first Sbox layer, given an arbitrary run, it is possible to obtain 3 (linear combinations of) PRG output bits by exploiting linear dependencies among the key bits used in the Share function and the inputs to the first Sbox layer (without performing any guesses). However, unlike the previous method, the indices of these 3 output bits vary according to the run data and it is not compatible with the attack described in Section 5.1.

Example 3. Let us focus of the first equation in (4),

$$w_{c_1}^{(3)} = (w_{a_2}^{(3)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_2}^{(1)} \cdot w_{a_3}^{(3)}) \oplus (w_{a_2}^{(3)} \cdot w_{a_3}^{(1)}) \oplus R_3(c_1) \oplus R_1(c_1).$$

If $w_{a_2}^{(3)} = w_{a_3}^{(3)} = 0$, then, we can calculate $R_1(c_1)$.

Example 4. As a slightly more complex example, if $w_{a_2}^{(3)} = 0, w_{a_3}^{(3)} = 1$, then we can calculate

$$w_{a_3}^{(1)} \oplus R_1(c_1) = w_{a_3} \oplus w_{a_3}^{(2)} \oplus w_{a_3}^{(3)} \oplus R_1(c_1). \tag{11}$$

On the other hand, recall from (5) that w_{a_3} can be expressed as a linear equation in the secret key, which is the sum of 3 shares $w_{a_3} = l_{a_3}(x_1) \oplus l_{a_3}(x_2) \oplus l_{a_3}(x_3)$. Plugging the value of w_{a_3} into (11) and we obtain

$$w_{a_3}^{(1)} \oplus R_1(c_1) = l_{a_3}(x_1) \oplus l_{a_3}(x_2) \oplus l_{a_3}(x_3) \oplus w_{a_3}^{(2)} \oplus w_{a_3}^{(3)} \oplus R_1(c_1).$$

As noted, the value of the left hand side is known, and so is the value of the right hand side. Moreover, the value of $l_{a_3}(x_2) \oplus l_{a_3}(x_3) \oplus w_{a_3}^{(2)} \oplus w_{a_3}^{(3)}$ is known, hence we deduce $l_{a_3}(x_1) \oplus R_1(c_1)$ which is a PRG output bit of P_1 (more precisely, it is a linear combination of output bits).

By exploiting the above extraction method, given a parameter t and a run, we can generate a $6t$-bit string where the $3t$-bit prefix specifies which 3 (linear combinations of) PRG output bits (of the unopened player) are derived for the first t Sboxes in the first layer of LowMC. The $3t$-bit suffix specifies the values of these bits for the run. This extraction method can be used to match an arbitrary PRG seed with the PRG outputs of the unopened player in 2^{6t} runs in time 2^{3t} (after inserting the runs into a hash table according to their $6t$-bit strings defined above). Note that since there are 2^{3t} possible equations, we have to match the PRG output generated with the given seed against 2^{3t} hash table entries, hence the time complexity is 2^{3t} rather than 1.

We do not base our attack on this extraction method since we can generally obtain a better time complexity with the first two. On the other hand, using this method has minimal memory complexity, as the runs are merely stored in a hash table. Hence, it can be used to obtain time-memory tradeoffs by combining it with the previous two methods.