Cryptanalysis of Curl-P and Other Attacks on the IOTA Cryptocurrency

Ethan Heilman[1], Neha Narula[2], Garrett Tanzer[3], James Lovejoy[2], Michael Colavita[3], Madars Virza[2], and Tadge Dryja[2]

[1] Boston University
[2] MIT Media Lab
[3] Harvard University

Abstract. We present attacks on the cryptography formerly used in the IOTA blockchain, including under certain conditions the ability to forge signatures. We developed practical attacks on IOTA's cryptographic hash function Curl-P-27, allowing us to quickly generate short colliding messages. These collisions work even for messages of the same length. Exploiting these weaknesses in Curl-P-27, we broke the EU-CMA security of the former IOTA Signature Scheme (ISS). Finally, we show that in a chosen-message setting we could forge signatures and multi-signatures of valid spending transactions (called bundles in IOTA).

Keywords: cryptocurrencies, signature forgeries, cryptographic hash functions, cryptanalysis

1 Introduction

Cryptocurrencies rely on digital signatures to prove a user's intent to transfer funds. For performance, in many signatures schemes a user signs the hash of a payment's message instead of the message itself. In these schemes, if the underlying hash function is compromised, it can become possible for an attacker to forge digital signatures on payments.

This paper presents attacks on the signature scheme used to authorize payments in a cryptocurrency known as IOTA. These attacks work by exploiting a cryptographic weakness in IOTA's hash function, Curl-P-27. Importantly, our attacks were disclosed and patched in August 2017, and thus no longer impact the security of IOTA's signature scheme [20].

IOTA is a cryptocurrency designed for use in the Internet of Things (IoT) and automotive ecosystems. As of July 24, 2018, it had a market capitalization of $2.6B US dollars, making it the 9th most valuable cryptocurrency in the world [9]. Many companies are partnering with IOTA, including automotive manufacturer Volkswagen and the large European industrial firm Bosch. IOTA issued an announcement that Volkswagen is planning to release a product using IOTA in early 2019 [7]. At the end of 2017, Bosch purchased "a significant amount of IOTA tokens" [6]. In addition, the IOTA Foundation signed an agreement

with the city of Taipei to use IOTA in its smart city initiatives, including its Digital Citizen Card project [8].

IOTA uses cryptographic signatures to authorize payments by users. The IOTA Signature Scheme (ISS) is based on Winternitz One-Time Signatures [25], but unlike traditional Winternitz, in IOTA users sign the hash of a message. Thus, the security of ISS relies on its cryptographic hash function, which was Curl-P-27. By applying common differential cryptanalysis methods, we are able to quickly create messages of the same length which hash to the same value with Curl-P-27, breaking the function's collision resistance. We find an upper bound on the average number of queries to Curl-P-27 to generate a collision.

Using this collision attack, we can generate signature forgeries in IOTA. Our attacks on the IOTA signature scheme function in a *chosen-message* setting, where an attacker creates two payments—a benign payment and a malicious payment—such that a signature on the benign payment is also a valid signature on the malicious payment. Our analysis is just on the IOTA signature scheme and does not include the security of the IOTA network as a whole. These attacks apply to both normal and multi-signature IOTA payments. Spending from a multi-signature address requires one user to produce a payment for another user to sign, which fits exactly in the chosen-message setting of our attack. We detail how to apply our attack to IOTA payments which spend from multi-signature addresses, and provide a tool for creating collisions in single-signature and multi-signature IOTA payments. We also evaluate the resources required to perform the attack, and show that using 80 cores, we can create colliding IOTA payments in less than twenty seconds on average.

1.1 Vulnerability Status and Impact

On July 14, 2017, some of the authors began a disclosure process with the IOTA developers. We negotiated a timeline for them to patch the vulnerability and a date after which we could publish our results. On August 7, 2017, the IOTA developers deployed a backwards-incompatible upgrade to mitigate this vulnerability by removing the use of Curl-P-27 to generate signatures in IOTA, and replacing it with another hash function [32]. In order to perform the upgrade, deposits and withdrawals were halted on Bitfinex for approximately three days [4]. All users who held IOTA directly (not via an exchange) were encouraged to upgrade their wallets and addresses. On September 7, 2017 we published our vulnerability report describing the nature of our attack [20].

Our vulnerability report included example Curl-P-27 collisions and signature forgeries on validly formatted IOTA payment messages, as well as software to validate these examples. In this paper, we expand upon our previous results by detailing the cryptanalytic techniques we used to break Curl-P-27's collision resistance and providing software to generate said signature forgeries. We also extend these techniques to develop an attack against IOTA's multi-signature scheme when Curl-P-27 is used—the multi-signature setting is particularly well-suited to chosen-message attacks. We did not send any of these forged signatures to the IOTA network or interfere in the IOTA network in any way. As Curl-P-27

is no longer used for ISS, the signature forgery attacks presented in this paper do not impact present-day IOTA. This includes our multi-signature attack in Section 5.2. Curl-P-27 is still used in other parts of IOTA [14]. We do not present attacks on these uses.

2 Related Work

After the release of our initial report, Colavita and Tanzer [10] independently reproduced and implemented our cryptanalysis, as well as proved some new results about Curl-P's round function—namely, that it is a permutation and that it diffuses differentials across rounds in accordance with a particular closed-form expression. They are now collaborating on this paper.

Differential cryptanalysis techniques were first published in 1991 by Biham and Shamir [3] (researchers at IBM had discovered similar techniques in 1974 but chose not to disclose them publicly [11]). In this paper, we present a very simple application of differential cryptanalysis on a balanced ternary cryptographic hash function. Apart from our initial vulnerability report and [10] we are aware of no prior research on the differential cryptanalysis of balanced ternary-based cryptographic hash functions. However, there is work designing and analyzing a ternary-based cryptographic pseudo-random sequence generator [16].

Exploiting our cryptanalysis of Curl-P-27, we present a chosen-message attack on ISS's unforgeability. Although the danger of broken collision resistance—and the chosen message attack model—may not be immediately apparent, we see a cautionary tale in work on the MD5 hash function. In 2004, Wang et al. released the first complete collision for MD5 [35], and soon after published a generic procedure for generating random collisions [34]. In 2005, Lenstra joined Wang to apply this cryptographic vulnerability to X.509 certificates, a cornerstone of the public key infrastructure that enables protocols like HTTPS, and was able to construct pairs of colliding certificates [24]. Amidst doubts that a certificate authority would sign such suspicious certificates, or that they would even be exploitable once issued because they lacked "meaningful" structure, Stevens joined Lenstra et al. in 2007 to extend the original random collision attack on MD5 to a chosen-prefix collision attack [30]. This work culminated in 2009, when Stevens et al. announced that they had managed to forge a X.509 certificate with certificate authority privileges that passed verification on all major browsers [31], causing vendors to immediately obsolete MD5.

In October 2017, after the IOTA developers transitioned from using Curl-P-27 to using Kerl—based on Keccak—as the hash function in the IOTA Signature Scheme, an unrelated vulnerability called the 13 or M attack was discovered [23]. This exploit relies on the fact that in IOTA's signature scheme—which signs the message's hash in chunks with values in $[-13, 13]$—a signature for the number 13 (also represented as 'M') reveals as plaintext a derivative of the private key that can be used to forge all subsequent chunks. The IOTA Foundation patched this vulnerability by requiring that if a message hash to be signed includes a 13, then the user must alter the message until no 13s are present in the digest.

As an additional remediation step, the IOTA developers transferred potentially compromised funds to addresses under its control, providing a process for users to later apply to the IOTA Foundation in order to reclaim their funds [27].

3 Background

In this section, we provide the necessary preliminaries to understand our attacks. We start with a short review of some of IOTA's uncommon design features and terminology. We also provide an overview of the Curl-P hash function and the IOTA Signature Scheme (ISS).

3.1 IOTA Design

IOTA currently has several uncommon design features. First, IOTA uses balanced ternary instead of binary; second, payments in IOTA are known as bundles; third, IOTA uses a a new data structure called a tangle rather than a traditional chain of blocks; and fourth, IOTA employs a trusted party called a coordinator to checkpoint state and approve payments.

IOTA's data structures use balanced ternary, or base three; instead of bits $\in \{0, 1\}$, it uses *trits* $\in \{-1, 0, 1\}$, and instead of bytes of eight bits, it uses *trytes* of three trits. A tryte is represented as an integer $\in [-13, 13]$. IOTA often serializes trytes as the letters *A*-*Z* and the number 9.

A payment in IOTA is represented by a data structure called a *bundle*. Bundles are composed of multiple transactions, but IOTA transactions are not like transactions in other cryptocurrencies; they are buffers which store inputs or outputs. IOTA transactions include, among others, address, signature, value, and tag fields. We provide an detailed description of the IOTA bundle and transaction format when describing our attacks in Section 5.

IOTA is built upon the concept of a *tangle* [26]. This is similar to a Directed Acyclic Graph-chain (DAG-chain), where each block can reference more than one block parent [29]. In IOTA's case, however, there are no blocks to aggregate multiple payments. Instead, each transaction must have a nonce proving Proof-of-Work (PoW) and include pointers to two other transactions. In order to add transactions to the tangle, a user selects two *tip* transactions from the tangle to reference in her transaction. Once created and signed, the user performs sufficient Proof-of-Work and broadcasts the transaction (or transactions, in the case of a bundle) to the IOTA network.

In IOTA as it is currently deployed, a bundle must also be approved by the coordinator to be accepted. The coordinator is a trusted party run by the IOTA developers that approves and checkpoints the state of the tangle by signing it. This has led to concerns that IOTA is centralized, or under the control of the IOTA developers [33]. The IOTA developers argue IOTA is not centralized and that the coordinator is a temporary measure . The source code for the coordinator is not publicly available. Since we did not interact with the IOTA network we cannot confirm how the coordinator would impact our proposed attacks, but

we are not aware of any mechanism in the coordinator that would prevent the attacks presented in this paper.

3.2 IOTA's Signature Scheme (ISS)

IOTA uses a signature scheme inspired by the Winternitz One-Time Signatures (W-OTS) [25]. W-OTS is an optimization of Lamport signatures [22], operating on multiple bits (in IOTA, trits) at once to trade computational cost for decreased public key size.

ISS differs in several important aspects from W-OTS. First, ISS operates on the hashes of messages rather than on the messages directly, as in traditional W-OTS. Second, rather than use a checksum, ISS performs a technique dubbed *normalization* on the hash of the message.

ISS has three security levels. The first security level only signs the first third of the message hash. The second security level signs the first two thirds of the message hash. Finally, the third security level signs the entire message hash. Because our attack works against the highest security level, it also works against any of the lower security levels. For this reason, when we talk about ISS we will implicitly assume that security level three is used.

3.3 Curl-P

In this section, we describe the Curl-P hash function. Curl-P (sometimes referred to as Curl) is a cryptographic hash function designed specifically for use in IOTA . It has been used for a number of purposes in IOTA, including creating transaction addresses, creating message digests, Proof-of-Work (PoW), and hash-based signatures. At a high level, Curl-P follows the pattern of a Sponge Construction [2, 18], but it differs in some key areas. As the IOTA project has not provided any formal specification or analysis of Curl-P, we base our description on the open source implementation of Curl-P made available by the IOTA developers.

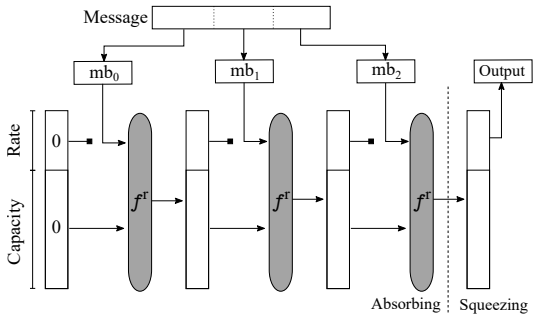

Fig. 1. The Curl-P construction.

Unlike most cryptographic hash functions, Curl-P operates on trits in balanced ternary. For clarity, we represent individual trits with lowercase letters such as a, b, c, x, y, z and sequences of trits as uppercase letters such as $\mathrm{S}, \mathrm{N}, \mathrm{X}, \mathrm{Y}$, unless we are referring a particular trit within a sequence of trits, where we will use subscript notation such as S_i. Following IOTA's convention, the R in Curl-P-R denotes the number of rounds used (*e.g.,* Curl-P-27 denotes 27-round Curl-P).

As shown in Figure 1, Curl-P operates as follows: (1) Curl-P initializes an all zero state S of length 729 trits. (2) The message is broken into message blocks $\mathsf{mb}_0 \cdots \mathsf{mb}_n$, each 243 trits. Curl-P employs no message padding; instead, if a message is not a multiple of 243 in length, the last message block is allowed to be less than 243 trits.[4] (3) In turn, each message block $\mathsf{mb}_0 \cdots \mathsf{mb}_n$ is copied into the first third of the state S and then that state S is transformed by the function f^{r}. (4) Finally, when no more message blocks remain Curl-P returns the first third of the final state as the hash output. For a more detailed description, see Algorithm 1.

Algorithm 1: The sponge-like construction used by Curl-P.

Function `CurlHash(msg):`
- $\mathsf{S} \leftarrow \{0\}^{729}$;
- **for** $p \leftarrow 0$; $p < |\mathsf{msg}|$; $p \leftarrow p + 243$ **do**
  - **if** $p + 243 < |\mathsf{msg}|$ **then**
    - $\mathsf{mb} \leftarrow \mathsf{msg}[p, p + 243]$;
  - **else**
    - $\mathsf{mb} \leftarrow \mathsf{msg}[p, |\mathsf{msg}| - 1]$;
  - $\mathsf{S}[0, |\mathsf{mb}|] \leftarrow \mathsf{mb}$;
  - $\mathsf{S} \leftarrow f^{\mathrm{r}}(\mathsf{S})$;
- **return** $\mathsf{S}[0, 243]$;

Now let's turn our attention to the function f^{r}, which is used to transform the state S. The transformation function f^{r} is actually just the function f recursively called on the state S for r rounds, *e.g.,* $f^3(\mathrm{S}) = f(f(f(\mathrm{S})))$. Curl-P-27 is the Curl-P hash function which uses f^{27} as its transformation function.

Each round of f^{r} generates a new state from the current state by calling f. As described in Algorithm 2, each trit in the new state is determined by applying the simple function g to a pair of trits in the current state. Each trit in the current state is used twice, once as the first parameter to g (represented by a) and once as the second parameter (represented by b). In Table 1, we give g as a substitution box or s-box.

[4] In some versions of Curl-P an error is thrown and hashing halted if the message length is not an even multiple of 243

$$c = g(a, b)$$

	$b = -1$	$b = 0$	$b = 1$
$a = -1$	1	1	-1
$a = 0$	0	-1	1
$a = 1$	-1	0	0

Table 1. S-box used by Curl-P: takes two trits a, b and returns a trit c.

Algorithm 2: The transform function $f(\mathrm{S})$ called by the Curl Hash Function.

```
i ← 0;
for pos ← 0; pos < 729; pos ← pos + 1 do
    j ← i;
    if i < 365 then
        i ← i + 364;
    else
        i ← i − 365;
    N[pos] ← g(S_j, S_i);
return N;
```

4 Cryptanalysis of Curl-P

In this section, we apply common differential cryptanalysis methods to engineer meaningful full-state collisions in Curl-P-27. Our attack constructs two messages of the same length which differ at only a single trit position and hash to the same value under Curl-P-27. Our technique lets us have a large degree of control over the content of the colliding messages, including arbitrary message prefixes and suffixes. In the next section, we will exploit this control over Curl-P-27 to forge signatures on valid IOTA payments.

We were unable to find a formal specification or documentation of Curl-P or Curl-P-27 beyond the source code published as part of the IOTA open source project. Furthermore, in our correspondence with the IOTA developers, they have stated that Curl-P-27 is designed to collide for specific sets of inputs [5]. In fact, Curl-P-27 is clearly non-random. As explored in detail by [10], Curl-P-27's non-random behavior can be observed in messages of the same length; collisions and second preimages are trivial to generate for messages of different lengths. Thus, to ensure we have truly broken Curl-P-27 we show that our collision attack meaningfully breaks a security property of Curl-P-27 on which the IOTA Signature Scheme (ISS) depends (see Section 5).

At a high level, our attack works as follows. We choose two messages of at least three message blocks in length which differ at only a single trit. To decrease the difficulty of our attack, we choose these messages such that they satisfy

certain constraint equations (explained in Section 4.2). Once we arrive at the message blocks that differ between the two messages we need to ensure that a collision occurs. To do this, we randomly flip a set of trits in both messages. This set of altered trits is limited to the differing message block in each of the two messages. The idea is that after running the transform function f^{27} on the differing message blocks, the only differences are in the first third of the resultant states.

$$f^{27}(\mathrm{S})[243, 729] = f^{27}(\mathrm{S}')[243, 729]$$

Because Curl-P replaces the first third of the state with the next message block, these differences are erased, causing a full state collision. We exploit the differential properties of Curl-P-27 to brute force a 1-trit difference for many of the rounds of the transformation function such that it is unlikely these differences will diffuse beyond the first third of the state by the final round. Finding two messages that maintain a 1-trit difference across a sufficent number of rounds to generate a collision is upper bounded by 7.6 million or $2^{22.87}$ queries to Curl-P-27.

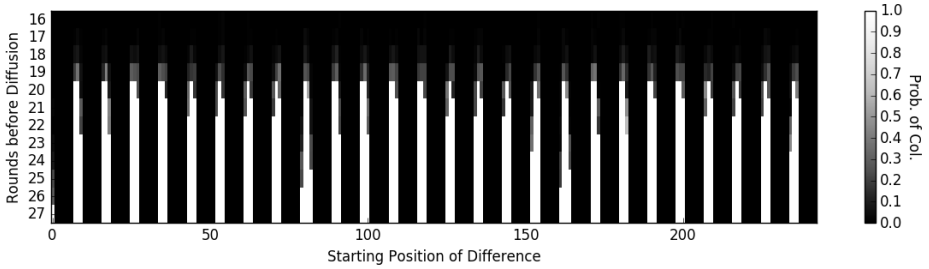

Fig. 2. Probability of a full state collision for a 1-trit difference at each position in a message block.

In Figure 2, we visualize how differences diffuse over rounds in Curl-P-27. We color the graph by the probability that a collision occurs given a particular starting position of a 1-trit difference (x-coordinate) and a lower bound on the number of rounds that a 1-trit difference is maintained (y-coordinate). To experimentally generate this dataset we performed 100 samples per position and round depth ($11 * 243 * 100 = 267300$ samples in total). Each sample was initialized to a random state with random difference injected at the anticipated position and round. For a more mathematical analysis see the recurrence used in [10].

Our collision attack uses a 1-trit difference at position 17 in the differing message blocks. Using the results of this experiment, we can calculate the probability of a collision if a 1-trit difference starting at position is maintained for certain number of rounds. For position 17, the probability of a collision is 1.0 for 20 rounds. Thus, if we prevent diffusion of a 1-trit difference starting at position 17 for at least 20 rounds we should find a collision. This attack should work for

some of the other positions in the input message block (as shown in [10]). Note that this is an upper bound on the amount of work to find a collision, since a collision could also occur from a 1-trit difference that is maintained for less than 20 rounds.

4.1 Differential Properties of Curl-P Transformation Function $f^{\mathbf{r}}$

In this section, we show how to find states that maintain a 1-trit difference for at least 20 rounds. This involves analyzing the differential properties of Curl-P's transformation function f.

Differential cryptanalysis is concerned with studying the propagation patterns of differences between two or more sets of inputs. The most common technique is the discovery of *differential trails*. A differential trail is a probabilistic bias of how a set of differences will propagate to another set of differences through many rounds of a cryptographic function. Here we only work with a specific differential trail, namely a 1-trit difference between the two states S,S$'$ under repeated applications of the transformation function f. We show that Curl-P has a strong bias toward maintaining a 1-trit difference across rounds (*i.e.,* applications of f).

Let's first introduce some necessary terminology. Since Curl-P operates on trits $\in \{-1, 0, 1\}$ instead of bits $\in \{0, 1\}$, we must use new notation for ternary differentials. To represent the difference between between two trits, x and x' we use $\ominus$ (*e.g.,* $0 \ominus -1$, which means either $x = 0$ and $x' = -1$ or $x = -1$ and $x' = 0$). By the term diffusion, we indicate that after an application of f the number of differences between the two states has increased (*i.e.,* the differences have diffused).

Our attack is built around the fact that the s-box g does not always propagate differences. For example, consider two sets of inputs and outputs to g: a, b, c and a', b', c' such that $g(a, b) = c$ and $g(a', b') = c'$. We make the following observations:

1. For all possible values, if $a \neq a'$ and $b = b'$ then it will always be the case that $c \neq c'$.
2. If $a = a'$ and $b \neq b'$ then both $c = c'$ and $c \neq c'$ are possible (*e.g.,* $a = a' = 1$, $b = 0$ and $b' = -1$ then $c = c' = 0$).

Each round of $f^{\mathbf{r}}$ *i.e.,* each application of $tranfs$ in $f^{\mathbf{r}}$, is called to update the state. As discussed in Section 3.3, each trit in the updated state depends on the output of applying g to two trits in the prior state. Each trit in the prior state can impact at most two trits in the updated state: once as the first variable a to the s-box g and once as the second variable b. This means that a 1-trit difference will always propagate to the next round, since when it is the first variable a to the s-box g, the output of g will differ based on a difference in a (as shown in part 1 of our observation). Thus if you apply f to two states S,S$'$ which have 1-trit difference the updated states f(S), f(S$'$) will either differ by 1 trit or 2 trits. It will never result in a 0-trit difference.

We model the probability that a 1-trit difference will remain a 1-trit difference across k rounds of Curl-P. As shown in Figure 3, by enumerating all possible inputs to g we develop a Markov model of the possible difference states after an application of f starting from a 1-trit difference. For instance, if the 1-trit difference in the current round is $0 \ominus 1$, then with probability 1/9 the difference stays the same (*i.e.,* $0 \ominus 1$) in the next round, with probability 2/9 the difference becomes $0 \ominus -1$ in the next round, or with 6/9 the number of differences increases from 1 to 2 (marked as the fail state as it fails to maintain a 1-trit difference).

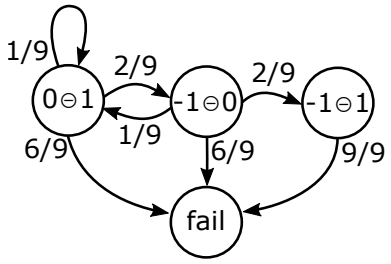

Fig. 3. The Markov chain represents the probability of starting from a single difference of a particular type and ending in a single difference.

As shown below, we convert the Markov model to a state transition matrix.

$$\begin{bmatrix} 1/9 & 2/9 & 0 & 6/9 \\ 1/9 & 0 & 2/9 & 6/9 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}^k$$

The top row represents the state transitions probabilities of $0 \ominus 1$, the second row $-1 \ominus 0$, third row $-1 \ominus 1$, and fourth row that a 1-trit difference diffuses to a 2-trit difference (the fail state). Using this matrix, we compute a lower bound on the probability that after k applications of f, the number of differences remains at 1. This is a lower bound as our analysis does not count transitions which increase to a difference of 2 trits or more and then later become a 1-trit difference.

Thus, starting from a 1-trit $0 \ominus 1$ difference we calculate a lower bound on the probability of a 1-trit difference by raising the matrix to the number of rounds we wish to investigate. For example, if we raise it to the power 3, the transition probabilities in the matrix represent the probability that you arrive at that difference after three rounds. Thus, we can measure the probability we don't fail after k rounds.

Earlier, we experimentally verified that if a 1-trit difference is maintained for 20 rounds of Curl-P-27 (*i.e.,* 20 applications of f), then the probability of a collision is 1.0. Using our state transition matrix we calculate for 20 rounds, our attack has a per query success probability lower bounded by 2^{-42}. That is,

we need to make on the order of 2^{42} queries to Curl-P-27 with different message pairs before finding a pair that will maintain a 1-trit difference for 20 rounds. This message pair will result in a collision. In the next section, we will show we can significantly reduce the necessary number of queries to Curl-P-27.

4.2 Solving for a 1-Trit Difference

In this section, we show how to reduce the number queries to Curl-P-27 by selecting messages with particular properties. We first show how to constrain two states S and S′, which differ by 1-trit, such that for at least 9 applications of f a 1-trit difference will be maintained (*i.e.,* there is no diffusion of differences). To do this, we represent f as a system of equations and solve for particular values of trits in states S and S′.

We can represent the transformation function $f^{\mathrm{r}}(\mathrm{S})$ as a series of equations. For example, a single call to f can be written as

$$f(\mathrm{S})_0 = g(\mathrm{S}_0, \mathrm{S}_{364}), f(\mathrm{S})_1 = g(\mathrm{S}_{364}, \mathrm{S}_{728}), \ldots, f(\mathrm{S})_{728} = g(\mathrm{S}_{365}, \mathrm{S}_0)$$

where $f(\mathrm{S})_0$ is the trit in position 0 of the updated state after f is applied. Since each round is just the recursive application of f, we can write the value of a particular trit after a number of rounds of f in terms of some of the initial values of the state S. We use superscript to denote the number of rounds of f. For example, with

$$f^2(\mathrm{S})_6 = g(g(\mathrm{S}_{366}, \mathrm{S}_1), g(\mathrm{S}_{184}, \mathrm{S}_{548}))$$

we specify the equation for the trit in position 6 after two rounds of f.

Using this representation, we find the equations guaranteeing that a 1-trit $0 \ominus 1$ difference is maintained for 9 rounds. We then find a message prefix that satisfies these equations. Our current process for finding this message prefix takes less than a second (see Section 5.3 for our performance evaluation). Additionally, given a particular message template, we only have to change a small set of trits in two message blocks to transform it into a satisfactory message.

4.3 Finding Collisions

We now combine our two methods to generate collisions for Curl-P-27. We refer to the technique shown in Section 4.2 of choosing a message prefix such that a 1-trit difference at a particular position will not diffuse across 9 rounds as the *constraint phase* of our attack. We call our method in Section 4.1 of trying different messages to increase the number of rounds for which a 1-trit difference is maintained the *brute-force phase*.

Our collision attack requires at least three message blocks: mb_a, mb_b, and mb_c, where mb_b contains the difference. Any number of message blocks can exist before mb_a. Any number of message blocks can exist between mb_a and mb_b. mb_c is always the next message after mb_b and overwrites the differences in the first third of the state created by mb_b. The actual value of mb_c has no impact on the attack and can be anything.

Our full attack works as follows. First, in the constraint phase of our attack we find a suitable message prefix by altering trits in parts of mb_a and mb_b, such that they guarantee a 1-trit difference across 9 rounds of f. Next, in the brute-force phase we randomly alter trits in particular positions in mb_b with the objective of finding two messages such that a 1-trit difference starting in position 17 is maintained across 20 rounds. As the constraint phase ensures that each attempt in the brute-force phase also maintains a 1-trit different across 9 rounds, the attack complexity of the brute-force phase is reduced from 20 rounds to 11 rounds. As a result, the lower bound of the success rate per query is reduced to approximately $2^{-22.87}$ or one out of 7.6 million.

As is typical in differential cryptanalysis, our probability calculations make the simplifying assumption that actual values of the differing inputs are uniformly random. Due to the low diffusion rate of Curl-P-27 and the non-random properties of Curl-P this assumption may not always hold. However, as seen in Section 5.3, the bounds given in this section are reasonably close to the actual results.

5 Exploiting Collisions in Curl-P to Forge Signatures

In this section, we show how our collision attack against Curl-P-27 can be used to perform a signature forgery attack against the IOTA Signature Scheme (ISS). Continuing from the previous section, we show how to create two valid IOTA bundles (*i.e.,* payments), which differ in at most two trits and have the same Curl-P-27 hash. Then, we will describe the setting of our attack which exploits these colliding bundles to forge signatures. Finally, we will show how to perform this attack against multi-signatures (multisigs) as used by ISS.

5.1 Chosen-Message Attack on ISS

Our attack is a chosen-message attack, which means that a malicious user Eve tricks a user Alice by asking Alice to sign a bundle, b_1, and then later producing a different bundle, b_2, which also verifies under the signature Alice provided. In more detail:

1. Alice generates a key pair $(\mathsf{PK}, \mathsf{SK})$.
2. Eve uses our collision attack on Curl-P-27 to produce two bundles b_1, b_2 such that $b_1 \neq b_2$ and $CurlHash(b_1) = CurlHash(b_2)$.
3. Eve sends b_1 to Alice and asks Alice to sign it. Alice inspects b_1 and confirms that it is benign.
4. Alice sends Eve a signature σ on b_1, *i.e.,* $\mathsf{Sign}(\mathsf{SK}, b_1) \to \sigma$.
5. Eve produces a signature, bundle pair (σ, b_2) such that $b_1 \neq b_2$, b_2 is a valid bundle, and b_2 verifies under Alice's PK even though Alice has never seen b_2.

In Section 4.3, we introduced the general format of our attack, which requires at least three message blocks mb_a, mb_b, and mb_c. To perform the first phase of the attack, we set certain trits in mb_a and mb_b to particular values. In the

brute-force phase, we change other trits in mb_b each attempt and check to see if we have achieved a collision. However, the bundles must pass the validity checks in the IOTA software in order for them to be accepted in IOTA as valid bundles, which limits the trits we can modify to perform our attack.

<table>
<tr><td colspan="5">Signature Fragment (6561)</td></tr>
<tr><td colspan="5">Address (243)</td></tr>
<tr><td>Value (81)</td><td>Tag (81)</td><td>TS (27)</td><td>Current (27)</td><td>Last (27)</td></tr>
<tr><td colspan="5">Bundle Hash (243)</td></tr>
<tr><td colspan="5">Trunk Transaction Hash (243)</td></tr>
<tr><td colspan="5">Branch Transaction Hash (243)</td></tr>
<tr><td colspan="5">Nonce (243)</td></tr>
</table>

Fig. 4. IOTA transaction format. Field sizes are in trits. The shaded fields are used to calculate the bundle hash of the bundle in which the transaction is included.

A bundle's hash is computed by hashing the concatenation of the address, value, tag, timestamp, current index, and last index fields of each transaction in the bundle. The format of a transaction is shown in Figure 4. Most of these fields are constrained in well-formatted bundles—for example, the values in a bundle cannot sum to a negative number, the timestamp must be within a certain range, and the indexes must line up with the transactions in the bundle. Tags do not impact the semantics or validity of the bundle and can contain arbitrary trits. Thus, for both the constraint phase and for each attempt in the brute-force phase of our attack, we only change the trits in the tags.

Another important question is where Eve can place the collision to cause damage. In our initial vulnerability report, we demonstrated colliding bundles for two different styles of attack: one which places the collision in the address field so Alice unwittingly signs a transaction which burns funds that were originally intended for Eve, allowing Eve to claim Alice made a mistake, and a second which places two collisions in two different value fields in a bundle so that Alice unwittingly signs a transaction which pays Eve more than intended. In the following section, we describe in detail the latter attack style for bundles which require multiple signatures, which fits our chosen message setting.

5.2 Multi-signature Attack

One criticism of the signature forgery attacks presented in our vulnerability report [20] is that they are chosen-message attacks, that is, Eve must ask Alice to sign a bundle. To help demonstrate the importance of chosen-message security, we now extend our attacks to the IOTA multi-signature (multisig) scheme [28]. In multisig, funds can be spent only by signatures from multiple parties. To spend, one party creates a bundle and asks the other party to sign it, which is exactly a

chosen-message attack. The IOTA Foundation encourages exchanges deploying a hot storage/cold storage solution[5] to use multisig for securely storing funds [12]. One of the main reasons multisig is used in a cryptocurrency context is that it requires that an attacker must compromise more than one party to steal funds. Our attack removes this security benefit of multisig. We will consider a simple case of a 2-of-2 multisig where two parties both sign to spend funds; however, our attack generalizes to more complex settings.

Consider two parties—Eve and Alice—each holding a pair of ISS keys—$(\mathsf{PK}_E, \mathsf{SK}_E)$ and $(\mathsf{PK}_A, \mathsf{SK}_A)$—and funds which can only be spent by both a signature from Eve's secret key and a signature from Alice's secret key. This implies that Eve and Alice previously entered into a 2-of-2 multisig and are now spending those funds jointly. Our attack will work as follows: Eve will compute two colliding bundles, one which pays funds to Alice and one of which pays funds to Eve . Eve will sign and send to Alice the bundle that pays Alice. Once she has Alice's signature, Eve will use it on the colliding bundle to create a valid bundle which Alice never saw or authorized, and will broadcast this bundle.[6] In this setting, Eve is either malicious or has been compromised by a malicious party.

Txn	**MB**	**Message block contents**				
0	0	Address				
	1	Value	Tag	TS	Current	Last
1	2	Address				
	3	Value	Tag	TS	Current	Last
2	4	Address				
	5	Value	Tag	TS	Current	Last
3	6	Address				
	7	Value	Tag	TS	Current	Last
4	8	Address				
	9	Value	Tag	TS	Current	Last
...						

Fig. 5. A bundle portioned into its message blocks. We target two collisions: One in the 17th position of message block 3 by manipulating the trits in the tag portions of message blocks 1 and 3, and one in the 17th position of message block 7 by manipulating the tags in 5 and 7. The red arrows indicate the collisions.

In order to construct such bundles, Eve places the collisions in certain value fields in certain transactions. Figure 5 shows the first four transactions in such a

[5] This is cryptocurrency terminology for structuring the control of an account holding funds such that any transfer of funds requires the consent of two secret keys. One of the secret keys is "cold," meaning it is kept in a location not connected to the internet such as an airgapped device.

[6] Note that if Alice is a cold wallet, she relies on Eve to broadcast the transaction.

bundle divided into message blocks. The highlighted fields are the trits relevant to our attack. Eve causes a collision in trit 17 of the value field in the second transaction (message block 3) by manipulating the trits in the tag fields both before and after the collision. By doing this, Eve can produce two bundles with different values in the second transaction that have the same bundle hash. Eve creates a second collision later on in the bundle in the fourth transaction (message block 7), this time arranging the collision so that the values still sum to zero in both colliding bundles. This serves to change what amounts are paid to whom in the transaction.

Generating these collisions essentially requires running the attack twice, sequentially. In our current collision tool, we require one transaction between the two transactions where we collide in the value fields. Other than this requirement, and the requirement that the collisions are not in the first or last transactions, we can handle bundles with different numbers of transactions. That our tool can only cause collisions in the 17th trit of a message block is a limitation of the tool's current implementation, not of the cryptanalysis techniques described in Section 4. Our tool does not depend on the specific addresses and values in the transactions to generate collisions, but the collision trits in the values that are changed must be different in order to produce valid bundles. For example, if trit 17 is zero in both Alice and Eve's output values in b_1, then flipping trit 17 in Eve's output to one in b_2 will cause the values in b_2 to not sum to zero. In b_1 Alice's output value's trit 17 should be one and Eve's should be zero.

In Appendix B we show the contents of two example bundles we created using this technique. In this example, the bundles are spending a multisig input of $500,000,000$ IOTA controlled by Alice and Eve. Alice signs a bundle which pays Eve 1 IOTA and the remainder to other addresses. In the colliding bundle, Eve receives $129,140,164$ IOTA, at the expense of Alice's address.

Generating colliding single-signature bundles operates in much the same way; our vulnerability report demonstrated a signature forgery on a bundle which paid out to three addresses. In the benign bundle b_1 Alice receives $50,000$ and $810,021,667$ IOTA to two addresses she controls and pays 100 IOTA to Eve. In the malicious bundle b_2 Eve changes this so that she receives $129,140,263$ instead of 100, at the expense of Alice's funds. We have not investigated the effects of placing collisions in fields besides the value and address. Other attacks might be possible.

5.3 Performance Analysis

We ran these attacks on a 80-core Intel machine with 8 2.4GHz 10-core Intel chips and 256 GB of RAM, running 64-bit Linux 4.9.74. Our attacks use all of the CPU but a negligible amount of the RAM. As described in Section 4.3, finding a collision consists of two phases: the constraint phase calculates the set of constraints, and the brute-force phase generates randomness in the tags to find collisions.

The constraint phase generates and solves eighteen equations, two for each of the first nine rounds of Curl-P-27. The constraint phase is implemented in

Part	Average	Min	Max
Constraint phase	1.1 s	0.27 s	6.1 s
Brute-force phase	7.2 s	0.04 s	74 s
Multisig	15.2 s	1.4 s	74 s

Table 2. Run time of the constraint phase, brute-force phase, and the entire multisig attack which involves running each phase twice. Measurements over 5000 iterations.

Python, and runs on a single core. We did not try to optimize the first phase. Table 2 shows the average, minimum, and maximum times of running the first phase 5000 times, when colliding on the 17th trit.

Table 2 also shows measurements for the brute-force phase, which uses the trits and template generated from the first phase to brute-force one collision. This is implemented in Go and parallelizes well, so we use all 80 cores of our server. On average it only takes 7.2 seconds to find one collision using the output of the first phase. On average, it takes 5.2M attempts to find one collision, with the minimum and maximum attempts over 5000 runs 1279 and 53M, respectively. This corroborates our analysis in Section 4.3.

In order to perform the multisig attack described in Section 5.2, we must run both the constraint phase and the brute-force phase twice, sequentially, to find two collisions. Using our collision tool, it takes on average 15.2 seconds to produce two multisig bundles which differ in two places. Table 2 shows the average, minimum, and maximum times for 5000 runs with the same starting bundle.

6 Discussion

The IOTA developers have made several statements discussing the impact and the cause of these vulnerabilities. We summarize these statements and address some of the concerns.

The IOTA developers have argued that our attack model is irrelevant in the context of the complete IOTA network: specifically, that the chosen-message setting is implausible because "in IOTA an attacker doesn't choose the signed message" [5]. In response to the critique on the choosen-message setting we extended our attacks to work on payments spending from a multisig address since the multisig protocol explicitly allows one user to choose the message that another user will sign.

The IOTA developers also argued that "even most valid attacks" would fail on the live IOTA network because of unspecified "protection mechanisms" in the closed-source coordinator [5, 13]. The attacks presented in the vulnerability report and this paper are against the IOTA Signature Scheme in isolation. We did not analyze these attacks within the context of the complete IOTA system.

Additionally, IOTA developers claimed that the ability to find colliding inputs to Curl-P-27 was intentional and was for the purposes of preventing "scam clones."

It is worth quoting them in full: "The IOTA team made a design decision early on to prevent this possibility [of scam clones] by purposefully introducing the Curl-P hashing function with known practical collisions. This had the express purpose of rendering fraudulent clones of the protocol useless in their application as a DLT protocol, while at the same time guaranteeing the security of the IOTA protocol and network as a whole." They argue that the closed-source IOTA coordinator would protect the IOTA network from these purposefully introduced flaws, which they refer to as a "copy-protection mechanism" [13]. Our research, in addition to discovering a novel attack on the IOTA Signature Scheme, may have uncovered an intentionally-placed backdoor in the cryptography of IOTA.

7 Conclusion

This paper presents chosen-message signature forgery attacks on the IOTA Signature Scheme when using the hash function Curl-P-27. We explain the cryptanalysis methods we used to create full-state collisions on same-length messages which differ in only a single position. We describe how to use these methods to create two valid IOTA bundles which can differ in multiple positions but still hash to the same value, and thus a signature for one is a valid signature for the other. We give examples placing these differences in the value fields of a bundle, and show that an attacker can produce such bundles using easily-accessible hardware in tens of seconds.

8 Acknowledgements

The authors would like to thank Andy Sellars, Weijia Gu, Rachael Walker, Joi Ito, Vincenzo Iozzo, Sharon Goldberg, and Ward Heilman for feedback and guidance.

References

[1] Mihir Bellare and Phillip Rogaway. "The exact security of digital signatures-How to sign with RSA and Rabin". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 399–416.

[2] Guido Bertoni et al. "On the indifferentiability of the sponge construction". In: *Lecture Notes in Computer Science* 4965 (2008), pp. 181–197.

[3] Eli Biham and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems". In: *Journal of CRYPTOLOGY* 4.1 (1991), pp. 3–72.

[4] Bitfinex. *IOTA Protocol Upgrade August 08, 2017*. `https://www.bitfinex.com/posts/215`, archived at `https://web.archive.org/web/20180722235151/https://www.bitfinex.com/posts/215`.

[5] Tangle blog. *Full Emails of Ethan Heilman and the Digital Currency Initiative with the IOTA Team Leaked*. http://www.tangleblog.com/wp-content/uploads/2018/02/letters.pdf, archived at https://web.archive.org/web/20180228182122/http://www.tangleblog.com/wp-content/uploads/2018/02/letters.pdf, https://archive.is/6imWR.

[6] Bosch. *Press release: Robert Bosch Venture Capital makes first investment in distributed ledger technology*. https://www.bosch-presse.de/pressportal/de/en/robert-bosch-venture-capital-makes-first-investment-in-distributed-ledger-technology-137411.html, archived at https://web.archive.org/web/20180724022550/https://www.bosch-presse.de/pressportal/de/en/robert-bosch-venture-capital-makes-first-investment-in-distributed-ledger-technology-137411.html.

[7] Crypt Briefing. *First VW IOTA Product Will Be Released Early Next Year*. https://cryptobriefing.com/vw-iota-product-released/, archived at https://web.archive.org/web/20180724021409/https://cryptobriefing.com/vw-iota-product-released/.

[8] Coindesk. *City of Taipei Confirms It's Testing IOTA Tech for ID*. https://www.coindesk.com/city-of-taipei-confirms-its-testing-iota-blockchain-for-id/.

[9] CoinmarketCap. *CoinmarketCap IOTA July 23 2018*. https://coinmarketcap.com/currencies/iota/, archived at https://web.archive.org/web/20180724020019/https://coinmarketcap.com/currencies/iota/.

[10] Michael Colavita and Garrett Tanzer. "A Cryptanalysis of IOTA's Curl Hash Function". In: (2018).

[11] Don Coppersmith. "The Data Encryption Standard (DES) and its strength against attacks". In: *IBM journal of research and development* 38.3 (1994), pp. 243–250.

[12] IOTA Foundation. *IOTA Guide – Generating Secure Multisig Addresses (hot and coldwallet)*. https://domschiener.gitbooks.io/iota-guide/content/exchange-guidelines/generating-multisignature-addresses.html, archived at https://archive.is/087kP.

[13] IOTA Foundation. *Official IOTA Foundation Response to the Digital Currency Initiative at the MIT Media LabPart 4 / 4*. https://blog.iota.org/official-iota-foundation-response-to-the-digital-currency-initiative-at-the-mit-media-lab-part-4-11fdccc9eb6d, archived at http://web.archive.org/web/20180727155405/https://blog.iota.org/official-iota-foundation-response-to-the-digital-currency-initiative-at-the-mit-media-lab-part-4-11fdccc9eb6d?gi=4be3ca82ed48.

[14] IOTAledger (github). *IOTA Kerl specification*. https://github.com/iotaledger/kerl/blob/master/IOTA-Kerl-spec.md, archived at https://web.archive.org/web/20180617175320/https://github.com/iotaledger/kerl/blob/master/IOTA-Kerl-spec.md. 2017.

[15] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Vol. 2. New York, NY, USA: Cambridge University Press, 2004.
[16] Guang Gong and Shaoquan Jiang. "The editing generator and its cryptanalysis". In: *International Journal of Wireless and Mobile Computing* 1.1 (2005), pp. 46–52.
[17] Leon Groot Bruinderink and Andreas Hülsing. ""Oops, I Did It Again" – Security of One-Time Signatures Under Two-Message Attacks". In: *Selected Areas in Cryptography – SAC 2017*. 2018, pp. 299–322.
[18] Bertoni Guido et al. *Cryptographic sponge functions*. 2011.
[19] Paul Handy. *Merged Kerl Implementation*. `https://github.com/iotaledger/iri/commit/539e413352a77b1db2042f46887e41d558f575e5`, archived at `https://archive.is/jCisX`.
[20] Ethan Heilman et al. *IOTA Vulnerability Report: Cryptanalysis of the Curl Hash Function Enabling Practical Signature Forgery Attacks on the IOTA Cryptocurrency*.
[21] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Second Edition. CRC Press, 2014.
[22] Leslie Lamport. *Constructing digital signatures from a one-way function*. Tech. rep. Technical Report CSL-98, SRI International Palo Alto, 1979.
[23] Willem Pinckaers (Lekkertech). *IOTA Signatures, Private Keys and Address Reuse?* `http://blog.lekkertech.net/blog/2018/03/07/iota-signatures/`, archived at `https://archive.is/CnydQ`. 2018.
[24] Arjen K. Lenstra, Xiaoyun Wang, and Benne de Weger. *Colliding X.509 Certificates*. Cryptology ePrint Archive, Report 2005/067. `https://eprint.iacr.org/2005/067`. 2005.
[25] Ralph C Merkle. "A certified digital signature". In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 218–238.
[26] Serguei Popov. "The tangle". In: *cit. on* (2016), p. 131.
[27] Ralf Rottmann. *IOTA Reclaim Identification Verification Process*. `https://blog.iota.org/iota-reclaim-identification-verification-process-e316647e06e6`, archived at `https://web.archive.org/web/20180710000243/https://blog.iota.org/iota-reclaim-identification-verification-process-e316647e06e6?gi=b8190e111e7f`.
[28] Dominik Schiener. *IOTA Multi-Signature Scheme*. `https://github.com/iotaledger/wiki/blob/master/multisigs.md`IOTA Multi-Signature Scheme. 2017 (accessed February 3, 2018).
[29] Yonatan Sompolinsky and Aviv Zohar. "Secure high-rate transaction processing in bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 507–527.
[30] Marc Stevens, Arjen Lenstra, and Benne de Weger. "Chosen-Prefix Collisions for MD5 and Colliding X. 509 Certificates for Different Identities". In: *Advances in Cryptology – EUROCRYPT 2007*. Springer. 2007, pp. 1–22.
[31] Marc Stevens et al. "Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate". In: *Advances in Cryptology – CRYPTO 2009*. Springer, 2009, pp. 55–69.

[32] David Snsteb. *Upgrades & Updates*. https://blog.iota.org/upgrades-updates-d12145e381eb, archived at https://web.archive.org/web/20180722232608/https://blog.iota.org/upgrades-updates-d12145e381eb?gi=51123f82db22.
[33] Eric Wall. *IOTA is centralized*. https://medium.com/@ercwl/iota-is-centralized-6289246e7b4d, archived at https://web.archive.org/web/20180616231657/https://medium.com/@ercwl/iota-is-centralized-6289246e7b4d. 2017.
[34] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions". In: *Advances in Cryptology – EUROCRYPT 2005*. Springer. 2005, pp. 19–35.
[35] Xiaoyun Wang et al. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, Report 2004/199. https://eprint.iacr.org/2004/199. 2004.

A Security Definitions

In this Section we provide a formal definition of the EU-CMA (Existential Unforgeability under Chosen Message Attack) and EU-RMA (Existential Unforgeability under Random Message Attack) for ISS (IOTA Signature Scheme). We use EU-RMA to model *known message attacks* where the known message is chosen at random. We discuss extending these definitions to include *chosen message attacks* which are limited to messages which are also valid IOTA bundles.

We briefly recall the standard definitions of digital signature schemes and their security, mirroring standard literature [15, 17, 21].

Definition 1. *A digital signature scheme is a triple of algorithms* (KeyGen, Sign, Verify) *working as follows:*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{PK}, \mathsf{SK})$*: On input* 1^λ*, the* key generator KeyGen *outputs a keypair* $(\mathsf{PK}, \mathsf{SK})$*, consisting of a public key* PK *and secret key* SK*.*
- $\mathsf{Sign}(\mathsf{SK}, \mathsf{msg}) \to \sigma$*: On input* SK *and message* msg*, the* signing algorithm Sign *outputs a signature* σ*.*
- $\mathsf{Verify}(\mathsf{PK}, \mathsf{msg}, \sigma) \to b$*: On input* PK*,* msg*,* σ*, the* verification algorithm Verify *outputs a decision bit* b*.*

We require perfect completeness*. That is, for any message* msg*, a validly generated signature must always be accepted:*

$$\Pr\left[\mathsf{Verify}(\mathsf{PK}, \mathsf{msg}, \sigma) = 1 \;\middle|\; \begin{array}{c}(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \mathsf{msg})\end{array}\right] = 1 \ .$$

The setting relevant for our work in Section 5 is a *chosen message attack* where, before outputting a forgery, an adversary gets to learn signatures for messages of his choice. In particular, for one-time signature schemes, the adversary is able to learn a signature for a single message. Other attack models include random message attacks (adversary is able to learn signatures on random message(s))

which we define to model a known message attack. Key-only attacks (adversary is only able to learn the public key). In particular, prior work offered a key-only attack of IOTA [23] succeeding with probability 1 for $\approx 3\%$ public keys. Our chosen message attack exploiting vulnerabilities in Curl-P-27 succeeds with probability 1 (under reasonable heuristics) for all public keys.

Definition 2. *A one-time signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is secure against chosen message attacks (or, EU-CMA secure), if no polynomial-time stateful adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *can output a fresh forgery, except with negligible probability:*

$$\Pr\left[\begin{array}{c}(\mathsf{msg} \neq \mathsf{msg}') \\ \textit{and} \\ \mathsf{Verify}(\mathsf{PK}, \mathsf{msg}', \sigma') = 1\end{array} \;\middle|\; \begin{array}{c}(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\mathsf{msg}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{PK}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \mathsf{msg}) \\ (\mathsf{msg}', \sigma') \leftarrow \mathcal{A}_2(\mathsf{st}, \mathsf{msg}, \sigma)\end{array}\right] \approx \mathsf{negl}(\lambda) \ .$$

Notably, ISS uses a hash-then-sign paradigm. Therefore, ability to find collisions for the underlying hash function directly yields to a chosen-message attack. Given two colliding inputs x_1 and x_2 ($\mathcal{H}(x_1) = \mathcal{H}(x_2)$), the adversary will first output $\mathsf{msg} = x_1$, and after receiving a correct signature $\sigma = \mathsf{Sign}(\mathsf{SK}, \mathsf{msg})$ from the challenger, will output $\mathsf{msg}' = x_2$ and $\sigma' = \sigma$. Because both messages yield the same value when hashed, $\mathsf{Verify}(\mathsf{PK}, \mathsf{msg}', \sigma')$ always accepts.

One could argue that the practical impact of not achieving EU-CMA security is hard to quantify. Indeed, if other parts of the overall system required messages to have certain structure, yet an attacker was only able to produce forgeries for messages that lack this requisite structure, these cryptographic breaks might not yield attacks exploitable against valid messages.

We rule out this possibility by devising a highly flexible collision-finding algorithm (see Section 5). In particular, our algorithm lets us freely generate collisions with arbitrary known prefixes and suffixes. This is sufficient, for example, to generate two colliding IOTA transactions, or two colliding multi-signature bundles. We explain how to generate such forgeries in Appendix B, and experimentally validate that the IOTA reference implementation from August 6, 2017, which is before the commit to change the hash function from Curl-P to Keccak [19], accepts the forged signatures.

Section **??** discusses *existential unforgeability under known message attack* (EU-KMA) against ISS. For completeness we now provide a security definition for EU-KMA which assumes that the known message is chosen randomly *i.e.,* sampled uniformly. For the sake of exactness we define this as a *random message attack* (EU-RMA). Note that for one-time signatures the adversary $\mathcal{A}$ is no longer stateful as it does not require any interacting with the security game beyond receiving $(\mathsf{PK}, \mathsf{msg}, \sigma)$ triple for randomly chosen msg, and outputting a forgery.

Definition 3. *A one-time signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is secure against random message attacks (or, EU-RMA secure), if no polynomial-time adversary* $\mathcal{A}$*, given a signature on a random message can output a fresh forgery, except*

with negligible probability:

$$\Pr\left[\begin{array}{c}(\mathsf{msg} \neq \mathsf{msg}') \\ \textit{and} \\ \mathsf{Verify}(\mathsf{PK}, \mathsf{msg}', \sigma') = 1\end{array} \;\middle|\; \begin{array}{c}(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{msg} \xleftarrow{\$} \{-1, 0, 1\}^{\mathrm{poly}(\lambda)} \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \mathsf{msg}) \\ (\mathsf{msg}', \sigma') \leftarrow \mathcal{A}(\mathsf{PK}, \mathsf{msg}, \sigma)\end{array}\right] \approx \mathsf{negl}(\lambda) \ .$$

The definitions given in this section are intended as a helpful guide for the security assumptions of ISS. Because these definitions are asymptotic definitions they do not cleanly map onto our practical attacks which deal with concrete computational resources. We hope that follow up research will extend this work to develop concrete definitions for security of ISS such as those given for RSA in [1].

B Example Colliding Bundles

Our previous vulnerability report detailed colliding bundles which execute the steal money and waste money attacks for single-signature bundles. In Section 5.2, we described the structure of these attacks and showed how we targeted collisions in value fields using the previous and following tags. Here, we describe in more detail example multi-signature bundles that have the same hash and spend different amounts to Alice and Eve.

Figure 6 describes two different bundles b_1 and b_2 that differ only in the value fields in two transactions. We give the address the funds are being spent to or from, the tags, and the values. Each bundle consists of 11 transactions: one input spending funds that were in a multisig address controlled by Alice and Eve (4), five outputs, including one change output, spending to Alice, Bob, Carol, Eve, and back to Alice and Eve's multisig address (0-3,10), and five extra transactions which are present only to hold signature fragments from Alice and Eve authorizing the spend (5-9). Signature-holding transactions have empty addresses and values. The tags in first four transactions are generated using our collision tool so that b_1 and b_2 will have the same hash.

Txn	**Address**	**Tag**	b_1 **Value**	b_2 **Value**
0	Bob	WJ9JPWIYIVQSTFNYY9HCZUQRVBK	182219672	182219672
1	Eve	CYBLAAX9ZA99Q9ZU9CXIU9DXCCW	1	129140164
2	Carol	GOUKGHTRFTLRHPOBZRMDLM9QIEM	400000	400000
3	Alice	FZZMZXCXWAI9SZAURCR9C9BXDCW	129140164	1
4	Alice,Eve	99999999999999999999999999	-500000000	-500000000
5		99999999999999999999999999	0	0
6		99999999999999999999999999	0	0
7		99999999999999999999999999	0	0
8		99999999999999999999999999	0	0
9		99999999999999999999999999	0	0
10	Alice,Eve	99999999999999999999999999	188240163	188240163

Fig. 6. An example multisig bundle spending from an address controlled by Alice and Eve. Transaction 4 is the funding transaction, and transaction 10 is the change which goes back to Alice and Eve. The collisions are in transactions 1 and 3. Transactions 5-9 are just to hold the signature fragments of Alice and Eve.