PING and REJECT:
The Impact of Side-Channels on Zcash Privacy

Florian Tramèr*
Stanford University

Dan Boneh
Stanford University

Kenneth G. Paterson
ETH Zürich

October 2, 2019

Abstract

We describe remote side-channel attacks on private (a.k.a. shielded) transactions in Zcash. Our attacks exploit a lack of isolation between the Zcash client's peer-to-peer and wallet functionalities, and weaken or break the anonymity of shielded transactions.

We describe two active attacks—REJECT and PING—on the implementation of in-band secret distribution in the main Zcash client. At a high-level, both attacks exploit differences in the way that a client processes transactions, depending on whether the client is the transaction's payee or not. The REJECT attack sends out specially malformed transactions that are accepted by all nodes except by the payee who responds with an explicit "reject" message. The PING attack relays a transaction to a node, immediately follows it with a "ping" message, and infers whether the node is the transaction's payee based on the timing of the ping's response.

These two types of attacks could enable an active remote attacker to:

1. Determine the identity of the payee for a shielded transaction being sent into the network.
2. Locate the Zcash node that holds the private incoming viewing key (ivk) that corresponds to an attacker-known shielded payment address ("zaddr").
3. Break Zcash's unlinkability of diversified payment addresses, by determining whether two attacker-known payment addresses correspond to a same private viewing key.
4. Corrupt and crash a Zcash node that holds the private ivk corresponding to an attacker-known shielded payment address.
5. Set up a remote timing side-channel on an ECDH key exchange between a victim node's private ivk and an ephemeral public key. In principle, this side-channel can be used to fully recover the victim's ivk, although the attack is probably impractical.

The vulnerabilities underlying these attacks affected Zcash's Sapling implementation, from the initial Sapling release (v2.0.0) until release v2.0.7-2. We have disclosed these attacks to Zcash's security team, who fixed them in the latest v2.0.7-3 release [1]. The Zcash team's security announcement can be found here [2]. Users who have updated to the latest release are no longer vulnerable to these attacks.

*Work performed while the first author was visiting ETH Zürich

1 FAQ

Who was affected? The PING attack could have affected any user of the official Zcash client (or of a Sapling-enabled Zcash fork) who has received shielded funds in a Sapling transaction.

The REJECT attack could have affected any user of the official Zcash client (or of a Sapling-enabled Zcash fork) who has generated a Sapling shielded payment address ("zaddr") and shared this address with one or more third parties (e.g., if you have given your zaddr to an exchange, posted it online somewhere, or otherwise shared it so that someone could send you shielded funds).

Users who upgraded to the v.2.0.7-3 Zcash release [1] are no longer affected by these attacks.

Sprout transactions were not affected by the REJECT attack and were likely not remotely exploitable by the PING attack.[1]

The attacks require an adversary to actively monitor the network's peer-to-peer traffic and inject its own messages. Thus, the attacks cannot be applied retroactively.

What should I do? If you use Zcash, update your client to the latest v.2.0.7-3 release [1, 2].

If you use a project that forked Zcash and that supports Sapling transactions, ensure that the vulnerabilities underlying these attacks have been fixed and upgrade your software.

Were my funds at risk? No! The attacks cannot be used to steal or counterfeit coins.

Was my privacy at risk? Yes! The attacks could have been exploited by an active adversary to tell whether you have received a shielded transaction, recover the IP address of your Zcash node, link together your diversified payment addresses, and (in principle) even to fully recover your incoming viewing key (an attacker in possession of your viewing key cannot steal your coins).

The attacks do not allow an attacker to decrypt memos or shielded amounts.

Have these attacks occurred? We do not know. The REJECT attack is targeted at a specific user, so the blockchain will show no evidence whether this attack has been mounted or not. To find out if you have been targeted by a REJECT attack, you have to check your Zcash client's logs to find evidence of "reject" messages being sent as a result of a malformed Note plaintext.

A user who has been subjected to the denial-of-service attack described in Section 4.3.4 would likely have noticed that their node crashed and that it could not be restarted.

The PING attack only sends valid messages to Zcash peers and thus leaves no suspicious trace. As the Zcash client logs only contain coarse timing information (each log-entry displays the date-time up to seconds), identifying an attack in the logs is probably not reliably possible.

What is the difference between REJECT and PING? Both attacks let an attacker tell whether a remote Zcash client was the payee of a particular shielded transaction.

REJECT only applies to shielded transactions specially (mal-)crafted by the attacker, while PING applies to any shielded transaction, even those sent by honest users.

The attacks differ in the side-channel that they exploit. REJECT causes the payee of a transaction to relay an explicit error message to the attacker, while PING exploits a timing side-channel wherein the payee of a transaction takes a larger amount of time to respond to a ping.

REJECT always succeeds, while the success rate of PING depends on the network jitter between the attacker and victim (the measurable timing difference is about 1 millisecond).

Was I safe if I used Tor? No. While Tor or other network anonymity tools can mitigate the threat of the IP-recovery attack described in Section 4.3.2, they offer no protection against linkability of diversified addresses (Section 4.3.3) or denial-of-service (Section 4.3.4).

Moreover, while the use of Tor might increase network jitter, it does not remove the timing side-channels which can be used to determine the payee of a shielded transaction (Section 4.3.1) or to recover an incoming viewing key (Section 4.3.5).

[1] The principle behind the PING attack technically also applies to Sprout, but the resulting timing side-channel is unlikely to be remotely exploitable.

2 Timeline

08.26.19 Our team discovered The REJECT attack.

09.13.19 We disclosed the REJECT attack to Zcash's security team (by encrypted email).

09.13.19 The Zcash security team discovered the PING attack.

09.15.19 Our team independently discovered the PING attack.

09.20.19 We disclosed the PING attack to Zcash's security team (by encrypted email).

09.24.19 The Zcash team released a fix for both attacks in version 2.0.7-3, along with a security announcement [2, 1].

09.28.19 Jonathan Leto reverse-engineered the bug underlying the REJECT attack from Zcash's fix, and described the resulting "IP address recovery" attack—one of the four variants of the REJECT attack that we discovered. Leto created CVE-2019-16930 to track the bug and the IP address recovery attack [3].

3 In-band Secret Distribution in Sapling

Notation and Preliminaries. Let q be the modulus of the pairing-friendly BLS12-381 curve. Let $\mathbb{J}$ be the group of points on Jubjub, Zcash's elliptic curve over $\mathbb{F}_q$ of order $8 \cdot r$, with r prime. Let $\mathbb{J}^{(r)}$ denote the prime-order subgroup of $\mathbb{J}$ of order r, and let $\mathbb{J}^{(r)*}$ be the set of points of order r (i.e., $\mathbb{J}^{(r)} \setminus \{\mathcal{O}\}$).

For clarity, and departing from the convention in the Zcash specification [4], we use lowercase symbols to denote scalars (e.g., ivk), and uppercase symbols to denote curve points (e.g., $\mathsf{PK_d}$).

Zcash's *diversified payment addresses* are standard static Diffie-Hellman public keys. The long-term private key is called an *incoming viewing key* (ivk) and the public key, given by $(\mathsf{G_d}, \mathsf{PK_d})$, is called a diversified payment address. They are computed as:[2]

GenIVK()	GenDiversified(ivk)
$\mathsf{ivk} \xleftarrow{R} \{0, \ldots, 2^{252} - 1\}$	$\mathsf{G_d} \xleftarrow{R} \mathbb{J}^{(r)*}$
return ivk	$\mathsf{PK_d} \leftarrow \mathsf{ivk} \cdot \mathsf{G_d}$
	return $(\mathsf{G_d}, \mathsf{PK_d})$

In-band Encryption. To send some value v to a shielded payment address $(\mathsf{G_d}, \mathsf{PK_d})$, a sender creates a new transaction with an *Output description* that contains, among other information, a *Note commitment*: $\mathsf{cm} = \text{Commit}(\mathsf{G_d}||\mathsf{PK_d}||v; \mathsf{rcm})$. In order to spend her coins, the receiver needs to prove that she knows an opening of the Note commitment cm.

Zcash's in-band secret distribution protocol enables the sender to securely transmit an opening of the Note commitment to the receiver as part of the transaction. For this, the sender performs a semi-static Diffie-Hellman key exchange involving the receiver's public parameters $(\mathsf{G_d}, \mathsf{PK_d})$ and a fresh ephemeral secret esk:

KeyAgree($\mathsf{G_d}, \mathsf{PK_d}$)
$\mathsf{esk} \xleftarrow{R} \mathbb{F}_r^*$
$\mathsf{EPK} \leftarrow \mathsf{esk} \cdot \mathsf{G_d}$
$\mathsf{SS} \leftarrow 8 \cdot \mathsf{esk} \cdot \mathsf{PK_d}$
$\mathsf{k} \leftarrow \text{KDF}(\mathsf{SS}, \mathsf{EPK})$
return $(\mathsf{k}, \mathsf{EPK})$

[2] Technically, $\mathsf{G_d}$ is computed using a standard "hash-and-check" approach that iteratively samples a random diversifier $d \xleftarrow{R} \{0, \ldots, 2^{88} - 1\}$, hashes d, and checks whether the result is a valid x-coordinate of a point in $\mathbb{J}^{(r)*}$.

The opening of the commitment cm is included in the *Note plaintext* (np). The sender encrypts np under k using a standard symmetric AEAD scheme, and appends the resulting ciphertext C as well as the ephemeral public key EPK to the transaction.

Blockchain Scanning. To recover coins sent to her, a Zcash user scans each transaction in the blockchain using her incoming viewing key ivk. Given a transaction containing an ephemeral public key EPK, a Note ciphertext C and a Note commitment cm, she computes:

$\text{TrialDecrypt}(\mathsf{ivk}, \mathsf{EPK}, C, \mathsf{cm})$

1: $\mathsf{SS} \leftarrow 8 \cdot \mathsf{ivk} \cdot \mathsf{EPK}$
2: $\mathsf{k} \leftarrow \text{KDF}(\mathsf{SS}, \mathsf{EPK})$
3: $\mathsf{np} = \text{Decrypt}_{\mathsf{k}}(C)$
4: **if** $\mathsf{np} = \perp$, **return** $\perp$
5: Parse np as $\mathsf{np} \coloneqq (\mathsf{G_d}, v, \mathsf{rcm}, \text{memo})$
6: $\mathsf{PK_d} \leftarrow \mathsf{ivk} \cdot \mathsf{G_d}$
7: **if** $\mathsf{cm} \neq \text{Commit}(\mathsf{G_d}||\mathsf{PK_d}||v; \mathsf{rcm})$, **return** $\perp$
8: **return** np

Unlinkable Diversified Addresses. The Zcash protocol enables a user to generate multiple diversified payment addresses linked to the same private incoming viewing key. That is, a user can generate multiple pairs $(\mathsf{G_d}^{(i)}, \mathsf{PK_d}^{(i)})$ satisfying the discrete-log relation $\mathsf{PK_d}^{(i)} = \mathsf{ivk} \cdot \mathsf{G_d}^{(i)}$. This enables a user to hand out a different public key to each entity that it wants to receive funds from, without requiring the creation of new accounts.

The unlinkability requirement for diversified addresses says, informally, that it should be hard to tell whether two different diversified payment addresses belong to the same user. The formal security requirement is defined by a game with a computationally bounded adversary $\mathcal{A}$:

$\mathsf{Unlinkability}_{\mathcal{A}}$

$b \xleftarrow{R} \{0,1\}$
$\mathsf{ivk}_0 \leftarrow \text{GenIVK}(), \mathsf{ivk}_1 \leftarrow \text{GenIVK}()$
$(\mathsf{G_d}^{(0)}, \mathsf{PK_d}^{(0)}) \leftarrow \text{GenDiversified}(\mathsf{ivk}_0), (\mathsf{G_d}^{(1)}, \mathsf{PK_d}^{(1)}) \leftarrow \text{GenDiversified}(\mathsf{ivk}_1)$
$(\mathsf{G_d}^{(2)}, \mathsf{PK_d}^{(2)}) \leftarrow \text{GenDiversified}(\mathsf{ivk}_b)$
$b' \leftarrow \mathcal{A}(\mathsf{G_d}^{(0)}, \mathsf{PK_d}^{(0)}, \mathsf{G_d}^{(1)}, \mathsf{PK_d}^{(1)}, \mathsf{G_d}^{(2)}, \mathsf{PK_d}^{(2)})$
return $b = b'$

The unlinkability of diversified addresses holds under the DDH assumption in $\mathbb{J}^{(r)*}$ [4].

4 The Attacks

At a high level, the goal of both the REJECT and PING attacks is for an adversary to tell whether a remote Zcash client succeeded in decrypting the Note ciphertext C of a particular shielded transaction. As the decryption only succeeds if the remote Node was the payee of the transaction, this information leaks to the attacker.

The two attacks differ in their setup (REJECT only applies to shielded transactions specially crafted by the attacker, while PING applies to any shielded transaction), and in the side-channel that they exploit (an explicit error message for REJECT, and a timing side-channel for PING).

Any references to source code files are from release v2.0.7 of the Zcash client which was still vulnerable to our attacks. The latest release v2.0.7-3 includes fixes for all the attacks that we report here.

4.1 The REJECT Attack

Upon seeing a new shielded transaction, the Zcash client attempts to decrypt the associated Note ciphertext C. If the decryption succeeds (i.e, this client is the transaction's recipient), the client then attempts to parse the Note plaintext `np` (line 5 in TrialDecrypt). A valid plaintext is a 564-bit string, where the first byte indicates the encoding version (`0x00` for Sprout and `0x01` for Sapling).

If the leading version byte takes on an incorrect value (e.g., other than `0x01` for a Sapling transaction), the plaintext serialization (defined in `Note.hpp`[3]) throws a `std::ios_base::failure` exception with error message "lead byte of SaplingNotePlaintext is not recognized".

The use of this exception-type for errors in message encoding is inherited from the Bitcoin client that Zcash is built upon. The exception is propagated all the way to the `ProcessMessages` method in `main.cpp`, where it is caught and causes a "reject" message to be sent back to the sender of the transaction.[4]

This provides the sender with an oracle indicating the successful decryption of a Note ciphertext with a specifically malformed plaintext (e.g., with a version byte of `0x02`).

We note that Zcash has an in-built anti-DoS mechanism, inherited from Bitcoin, wherein peers that send malformed transactions get assigned penalty scores and are ultimately dropped from the P2P network. Surprisingly, encoding errors such as this one (or others, such as supplying incorrect transaction flags) do not seem to trigger this anti-DoS mechanism.

The vulnerability does not seem to affect Zcash's implementation of the original Sprout version. While the serialization of Sprout Note plaintexts also throws an exception if the lead version byte is incorrect,[5] this exception is correctly caught during blockchain scanning.[6]

4.2 The PING Attack

After a Zcash client successfully decrypts a Note ciphertext and parses the Note plaintext, it checks that it has received a valid opening of the Note commitment `cm` (lines 6-7 in TrialDecrypt).

This involves the computation of a Pedersen hash function (see [4] for details) and two multiplications of a Jubjub point by a random 256-bit scalar. As these are costly cryptographic operations, a call to TrialDecrypt where the decryption of the Note ciphertext succeeds will take significantly longer than a call where the decryption fails (the time difference is on the order of a millisecond on a standard desktop machine).

A remote attacker can create a timing side-channel on the duration of the TrialDecrypt call by relaying a shielded transaction immediately followed by a "ping" message to a Zcash node. Zcash's peer-to-peer interface processes incoming messages in a serial fashion. It will thus first process the transaction, including the TrialDecrypt call, and then respond to the "ping" message. The time elapsed before the receipt of the "ping" response leaks information about the success of the Note decryption, and therefore on whether the node was the payee of the relayed transaction.

The attack applies even for transactions sent by honest users. The attacker would relay the transaction to other nodes and then send a "ping" to each recipient.

In principle, a similar timing side-channel exists in Zcash's older Sprout protocol. However, Note commitments in Sprout were hash-based and require a single SHA-256 computation [4]. Thus, the timing difference between a successful or unsuccessful call to TrialDecrypt is very low (sub-microsecond) and unlikely to be reliably measurable remotely.

4.3 Attack Consequences

4.3.1 Determining if a Node is the Payee of any Shielded Transaction

This attack on transaction anonymity applies to arbitrary shielded transactions. It is a straightforward consequence of the PING attack.

[3] `https://github.com/zcash/zcash/blob/0512e9e/src/zcash/Note.hpp#L157`
[4] `https://github.com/zcash/zcash/blob/0512e9e/src/main.cpp#L6366`
[5] `https://github.com/zcash/zcash/blob/0512e9e/src/zcash/Note.hpp#L98`
[6] `https://github.com/zcash/zcash/blob/0512e9e/src/wallet/wallet.cpp#L1806`

First, the attacker builds a timing baseline by running the ping attack multiple times against a target node, using shielded transactions that are explicitly not addressed to that node (e.g., the attacker can send shielded values intended to itself). The timing of the ping responses tells the attacker how long an unsuccessful TrialDecrypt call takes for that node.

Then, upon seeing a new shielded transaction, the attacker relays the transaction and a ping to the victim node. The attacker records the time elapsed before receiving the ping response, and checks whether there is a large difference from the previously measured baseline response time. If the response time is longer than expected, the node was probably the transaction's recipient.

The attacker can of course build baselines and relay transactions to multiple (or all) nodes in the network, thereby guessing the payee of all shielded transactions.

On a standard desktop machine, the timing difference between a successful and unsuccessful TrialDecrypt call is about 1 millisecond—a significant delay even over a noisy network connection. We built a proof-of-concept of this attack in a LAN, where the attacker can determine whether a node received a transaction with 100% precision (see Section 5). More generally, the success of the attack depends on the network jitter between the attacker and victim nodes.

4.3.2 Linking a Payment Address to a Zcash Node ("IP address Recovery")

Given a shielded payment address $(\mathsf{G_d}, \mathsf{PK_d})$, the REJECT and PING attacks can be used to determine which Zcash client this address belongs to. We describe the simpler REJECT variant.

The attacker builds a Note plaintext with an incorrect leading byte, encrypts it under $k = \text{KeyAgree}(\mathsf{G_d}, \mathsf{PK_d})$, and sends a transaction containing the Note ciphertext to all nodes in the P2P network to check which node responds with a "reject" message.

Unless the victim has taken measures to conceal the IP address of its Zcash node (e.g., by connecting via Tor [5]), either attack variant can link a payment address to a specific IP address.

One could argue that an attacker in possession of a victim's payment address could already have determined the victim's IP address when it originally received the payment address. But this need not always be the case. For example:

- The victim may have used an anonymization system like Tor when sharing its payment address with the attacker, but may not be hiding the IP address of the Zcash node that contains its private key.
- An attacker could obtain payment addresses through other means, e.g., by hacking a Zcash-supporting service that the victim was using.

4.3.3 Linking of Diversified Keys

Given two payment addresses $(\mathsf{G_d}, \mathsf{PK_d})$, $(\mathsf{G_d}', \mathsf{PK_d}')$, an attacker can determine whether they belong to the same user, thus breaking the strong unlikability guarantee for Zcash's diversified payment addresses.[7]

The attack simply consists in running the above "node identification" attack twice, with two transactions containing a Note plaintext encrypted for both payment addresses. The two payment addresses are derived from the same ivk if and only if the same node rejects both transactions.

This assumes that the victim's Zcash node contains a single incoming viewing key. If a user has generated multiple distinct Zcash accounts, the attack determines whether two payment addresses correspond to incoming viewing keys loaded in the same Zcash wallet (and thus belonging to the same user).

Note that this attack works even if the victim's Zcash client makes use of Tor or similar anonymization tools. The attacker simply sends the two transactions over the same Tor connection, while being oblivious to the actual IP address of the destination.

[7] The attack does not formally break the unlinkability security notion captured by the $\mathsf{Unlinkability}$ game, as the game does not provide the adversary $\mathcal{A}$ with a decryption-success oracle for malformed plaintexts.

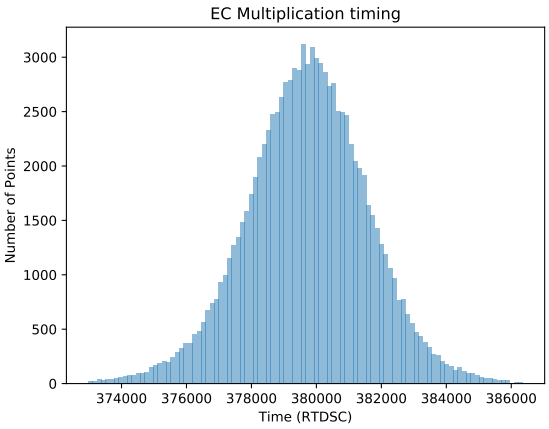

Figure 1: **Time to compute** $\mathsf{ivk} \cdot P$ **for a fixed** ivk **and one million random curve points** $P \in \mathbb{J}^{(r)*}$**.**

4.3.4 Disconnecting a Zcash Node

This is a curious additional consequence of the REJECT attack. Once a transaction containing a malformed Note plaintext is included in a mined block, the client with the corresponding ivk crashes when attempting to validate the block. The node logs the following error:[8]

```
zcashd: main.cpp:5042: void CheckBlockIndex(const Consensus::Params&):
    Assertion 'setBlockIndexCandidates.count(pindex) == 0' failed
```

This failure is pernicious. Even if the Zcash client is manually restarted, it re-crashes immediately with the same failed assertion when re-attempting to validate the block.

If an attacker were to get hold of shielded payment addresses for a large number of Zcash users, this failure could lead to a strong DoS attack on the network. Worse, if many Zcash miners had associated shielded payment addresses, such a DoS attack could be exploited to stifle the network's overall mining power (e.g., in preparation for a 51% attack or to remove mining competition).

One way to simultaneously affect all users with a shielded address would be to send a malformed ciphertext encrypted under the key $k = \mathrm{KDF}(\mathcal{O}, \mathcal{O})$ by setting $\mathsf{EPK} = \mathcal{O}$. However, this attack fails because the transaction validation routine—which occurs before the trial decryption—checks that EPK is not a low order curve point.

4.3.5 A Remote Timing Side-Channel on ECDH

Finally, both the REJECT and PING attacks can—in principle—be used to launch a remote timing attack on Zcash's implementation of the ECDH key exchange, in particular the Elliptic curve multiplication $\mathsf{ivk} \cdot \mathsf{EPK}$ performed in TrialDecrypt (line 1).

We note that the Zcash team was aware that the ECDH key exchange implementation is not constant time, and that this might be exploitable by a *co-located* adversary [6]. The key differentiator of the attack presented here is that REJECT or PING enable this timing side-channel to be used remotely.

The Elliptic Curve multiplication routine used in Zcash is indeed not constant-time: it uses a standard double-and-add procedure, with different (unified) formulas for point addition and doubling. The underlying $\mathbb{F}_q$ field arithmetic is not constant time as all operations end with a final reduction that subtracts the modulus q if necessary.

As a proof-of-concept, we adapted Kocher's original RSA timing attack [7] to Zcash's Elliptic Curve multiplication routine. For a fixed secret ivk, we locally timed the multiplication routine

[8] `https://github.com/zcash/zcash/blob/0512e9e/src/main.cpp#L5042`

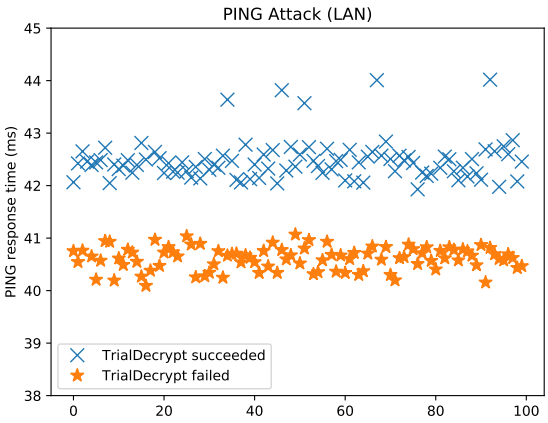

Figure 2: **Response times for the PING attack in a LAN setting.**

for 1 million random EPK points. The timing distribution is plotted in Figure 1, and is clearly not constant. Assuming we have already recovered the j most significant bits of ivk, we recover the $(j+1)$-th bit by correlating the time of a point doubling or point multiplication with the total multiplication time. Conditioned on all previous bits being recovered successfully, the following bit is recovered with 98.4% probability. With a clever backtracking mechanism to resolve the few false guesses, the full key can be recovered with about one million samples.

The query complexity of this attack is fairly high. The attack was performed in an "idealized" setting that ignores the time taken by the network and transaction verification, which would add significant noise and further increase the sample complexity of a full remote attack.

Our proof-of-concept of course also confirms the Zcash team's suspicion that timing side-channels could be exploited by a co-located adversary to recover secret key material.

5 Discussion

5.1 Attack Practicality

Payee Discovery. We built a proof-of-concept of the payee-discovery attack from Section 4.3.1 on a regtest network, where the attacker and victim machines reside in the same LAN. The attacker relays 200 transactions to the victim, half of which are payments to the victim. Figure 2 shows the distribution of the victim's response time to the ping message. Except for five clear outliers (with a response time of over 60ms), the response time indicates whether the victim was the transaction's payee with 100% precision.

The average timing difference of about 2ms between a successful or failed TrialDecrypt is significant, and can result in a fairly powerful attack even in a regular WAN, although the attacker will not be able to determine the payee of every shielded transaction with absolute certainty.

Linkability. We validated the REJECT variant of the node linkability attack from Section 4.3.2 and the diversified address linkability attack from Section 4.3.3 in a local Zcash network, wherein an attacker sends ciphertexts corresponding to malformed plaintexts to a victim node running the unmodified Zcash client.

Both attacks are very easy to mount. They simply require sending one or two transactions to every peer in the network and to check for an answer with a "reject" message.

One potential complication is that a peer that receives a malformed transaction could relay the transaction to the intended victim before the attacker's message reaches the victim. In this case, the victim will send a "reject" message to the relaying peer, and not respond to the attacker's message as the transaction is already in the victim's memory pool. Since a peer will first validate

a transaction before relaying it, the attacker's messages are likely to reach the victim node first. In the event that the attacker does fail to receive a "reject" message, the attack can simply be repeated with a different transaction.

The attack is cheap: the attacker's transaction can simply send 0 funds to the victim. The attack also works best when setting a fee of 0, as this limits the possibility of the transaction being mined—thereby crashing the victim node (we note that the victim node crashing as a result of a mined malformed transaction can also lead to an IP address recovery simply by monitoring which node disconnected from the network).

Denial of Service. We also validated the DoS vulnerability from Section 4.3.4 in a local Zcash network. Mounting a large-scale DoS attack would require an attacker to be in possession of shielded payment addresses for many Zcash users. A plausible candidate would be a service with many Zcash clients, such as an exchange that supports shielded addresses. A DoS attack on a large fraction of Zcash miners might be unrealistic as many miners may not have generated a shielded address.

Remote Timing. The feasibility of the remote timing attack on the ECDH key exchange is harder to assess. Our local timing attack on the elliptic curve multiplication routine required one million samples to be effective and this number would be much larger if the variability in network delay and transaction confirmation time were considered. Yet, it is not inconceivable that our timing attack could be improved further. If a remote timing attack were possible with a moderately large number of samples (e.g., 10,000), one difficulty with the REJECT attack variant might be in sending all those transactions to a victim before any one of them gets mined—thereby crashing the victim. We could not find statistics on how frequently transactions with a 0 fee get mined in Zcash. Another possibility would be for an attacker to eclipse a victim node from the rest of the network while performing the remote timing attack.

5.2 What is Affected

- The attacks affects the official Zcash client implementation between versions v.2.0.0 (the first Sapling version) and version 2.0.7-2. Any user of the client that has generated a shielded address and shared the public payment address ("zaddr") with a third-party may be vulnerable to the attacks described above (the IP recovery attack can be mitigated by using Tor). **All attacks have been mitigated in Zcash release v.2.0.7-3 [2, 1]. The attacks cannot be applied retroactively to earlier transactions.**
- Any Zcash forks with support for Sapling shielded addresses inherit these attacks and should update to Zcash's v.2.0.7-3 release or implement their own fix.
- All Zcash wallets that support shielded addresses and that use the official Zcash client are vulnerable. These include Zepio, ZecWallet and ZECmate Swing Wallet.
- Although our attacks do not exploit a specific flaw in the Zcash specification [4], we note that the specification fails to specify what action should be taken if the version byte of the Note plaintext is incorrect. This shortcoming in the specification had been previously noted.[9]

5.3 What is not Affected

- The not-yet-released Android Guarda wallet[10] with support for shielded transactions is immune to the REJECT attack as it simply ignores malformed Note plaintexts. The wallet is also immune to the PING attack since it does not act as a peer-to-peer node.

[9] `https://github.com/zcash/zips/issues/228`
[10] `https://github.com/guardaco/guarda-android-wallets/tree/gh-zapling-main`

- Blockchain scanning is not yet implemented in the currently developed Zebra client[11] so that client is not vulnerable to these attacks. As the Rust language does not support exceptions, it is plausible that the REJECT attack would not have affected Zebra.
- The `librustzcash` library used in the Zcash client has an additional implementation of Note decryption in Rust which returns `None` if the underlying plaintext is malformed or if the opening of the Note commitment is incorrect. As far as we can tell, this implementation is currently unused.

5.4 Remediations

The remediation for the REJECT attack is very simple: treat a plaintext parsing failure similarly to a decryption failure, and ignore the offending ciphertext. This fix was implemented in release 2.0.7-3 of the Zcash client [1, 2].

More generally, we believe that a more granular and rigorous treatment of error conditions (which appears to be a conscious design decision in Zebra) will go a long way to reduce the possibility of such bugs. We have investigated other situations where the Zcash client throws exceptions when failing to serialize messages (e.g., if the `overWintered` flag takes on an incorrect value), but have not found these to cause any vulnerabilities.

The REJECT and PING attacks exploit a coupling between the Zcash node's peer-to-peer functionality and its wallet functionality. Release 2.0.7-3 of the Zcash client [1, 2] addresses this issue by refactoring the TrialDecrypt functionality into a separate thread, that periodically pulls the list of recently received transactions and attempts to decrypt all associated Note ciphertexts.[12] The timing of the TrialDecrypt call thus no longer affects the timing of peer-to-peer functionalities such as ping messages—thereby preventing the PING attack.

Finally, we believe Zcash should strive towards producing a side-channel resistant implementation of their core cryptographic primitives. Side-channel resistance may have seemed like a secondary concern, given that the Zcash protocol is primarily non-interactive. As our attacks have shown, a single bug in the in-band secret distribution routine inadvertently allowed for a two-way interaction between an attacker and victim, thereby opening up a potential remote timing side-channel.

6 Acknowledgments

We thank the Zcash security team—particularly Sean Bowe and Benjamin Winston—for the responsible disclosure process, their clear and professional correspondence with us, and their rapid response to this vulnerability.

References

[1] Electric Coin Company, "Zcash release v2.0.7-3." `https://github.com/zcash/zcash/releases/tag/v2.0.7-3`, 2019.

[2] Electric Coin Company, "Security announcement 2019-09-24." `https://z.cash/support/security/announcements/security-announcement-2019-09-24/`, 2019.

[3] J. Leto, "Zcash metadata leakage CVE-2019-16930." `http://duke.leto.net/2019/10/01/zcash-metadata-leakage-cve-2019-16930.html`, 2019.

[4] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," tech. rep., Electric Coin Company, 2019. Version 2019.0.1 `https://github.com/zcash/zips/blob/d39ed0/protocol/protocol.pdf`.

[11] `https://github.com/ZcashFoundation/zebra`

[12] `https://github.com/zcash/zcash/blob/961c0d5/src/init.cpp#L664`

[5] Electric Coin Company, "Zcash documentation—Tor support in Zcash." `https://zcash.readthedocs.io/en/latest/rtd_pages/tor.html`, 2019. Revision fe830a5a.

[6] Electric Coin Company, "Zcash documentation—security warnings—side-channel attacks." `https://zcash.readthedocs.io/en/latest/rtd_pages/security_warnings.html#side-channel-attacks`, 2019. Revision fe830a5a.

[7] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Annual International Cryptology Conference*, pp. 104–113, Springer, 1996.