black hat USA 2023

AUGUST 9-10, 2023

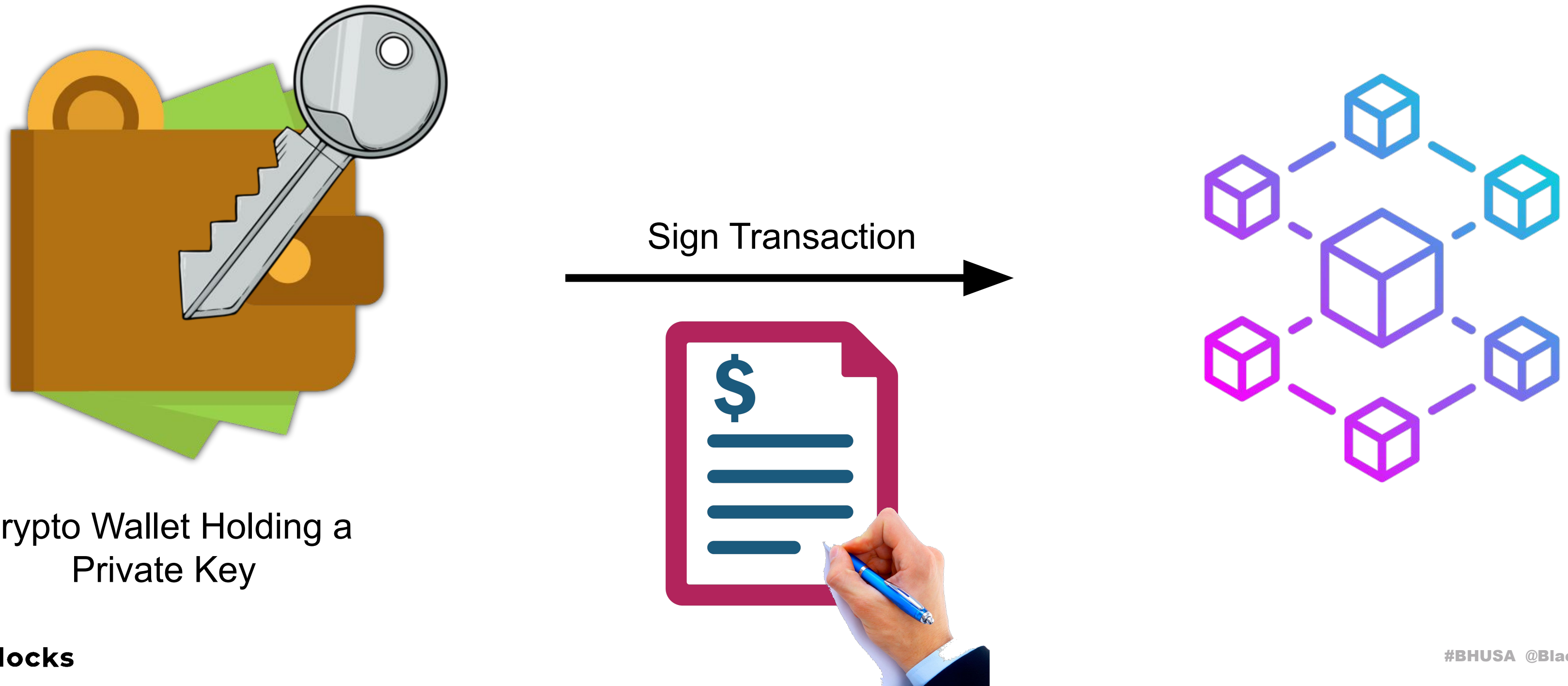
BRIEFINGS

Small Leaks, Billions Of Dollars: Practical Cryptographic Exploits That Undermine Leading Crypto Wallets

Speakers:
Nikolaos Makriyannis
Oren Yomtov

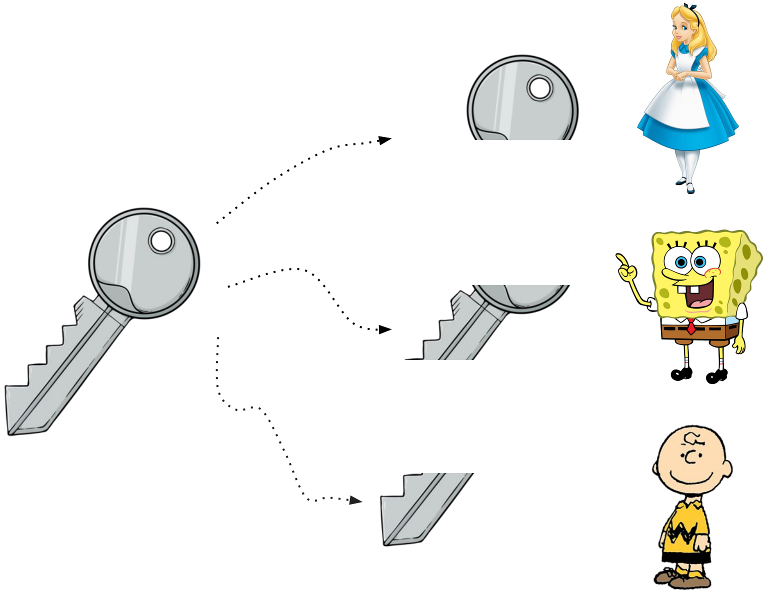
Intro to crypto wallets

Cryptocurrency Wallets 101

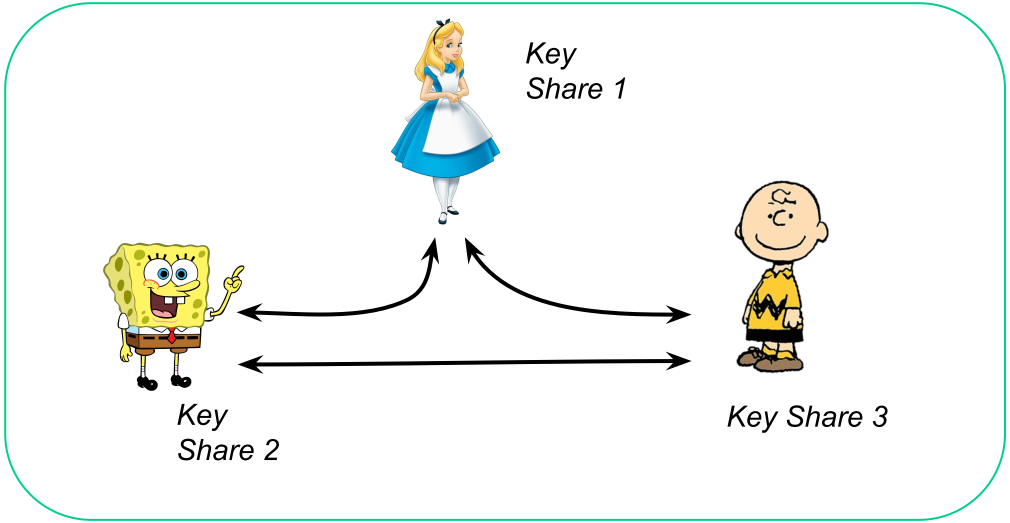


Crypto Wallet Holding a Private Key

What is MPC? (through the lense of threshold signing)



What is MPC?
(through the lense of threshold signing)



Generate public key and calculate signatures via an **interactive protocol**

The private key is ***NEVER*** *assembled in one place*

Small aside: MPC is much bigger than threshold signatures

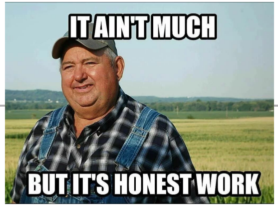
MPC (Multi-Party Computation) is the crown jewel of modern cryptography

Anything solved by trusting a centralized party can be solved trustlessly with MPC

Auctions *Games of Chance* *Nuclear Deterrence*

Voting *Threshold Signatures* *Private ML*

MPC is invented — 80s



Danish Sugar Beet Auction — 2008

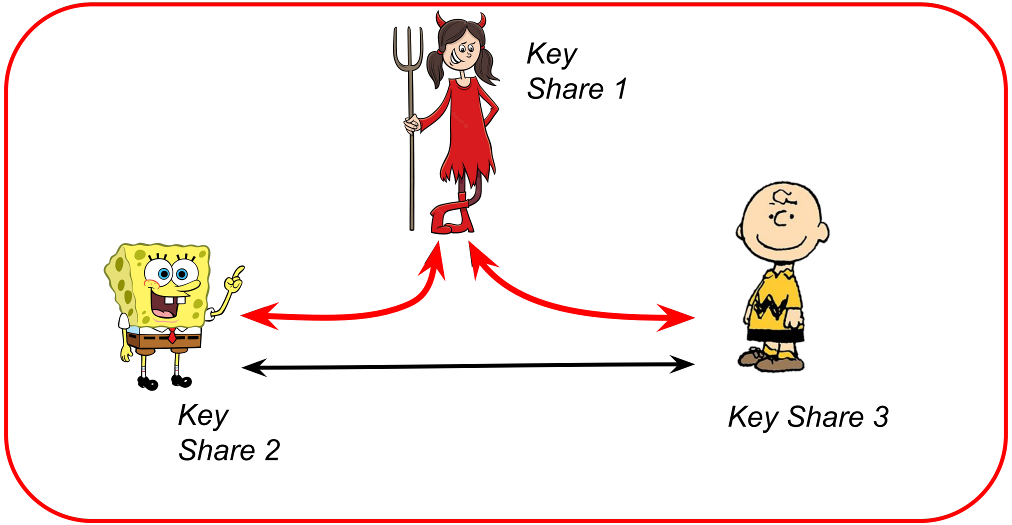
MPC Wallets — Late 10s

MPC Wallet Attack Outcomes

- Denial of Service
- Signature Forgery
- Private Key Exfiltration

Today's Talk

MPC Threat model



Malicious Alice wants to exfiltrate her counterparties' shares

Our Research Findings

Affected Parties

- Some of the biggest crypto exchanges (e.g. Coinbase WaaS)
- A number of crypto custodians (e.g. BitGo TSS)
- The most popular consumer MPC wallet (e.g. Zengo)
- Some of the most popular open source MPC libraries (e.g. Binance, Apache)

Our Findings

- Discovered 4 **novel attacks** (including **three 0-day**)
- Affecting **16** vendors / libraries
- Releasing 4 fully working **PoC exploits**
- Exfiltrated keys from 2 vendor **production environments**
- Most of our attacks are **not** implementation specific

The 3 attacks we'll be covering today

1. The most popular two-party signing protocol: Lindell17 (**high interactivity**)
2. The most popular multi-party signing protocols: GG18&20 (**med interactivity**)
3. A DIY protocol used by a crypto custodian: BitGo TSS (**low interactivity**)

Cryptographic exploit development

Math Background

- **We assume no familiarity** with advanced mathematics
- **Nothing** about elliptic curves (or even abstract groups)
- The **modulo** operator

$$x \,\%\, N$$

Remainder of
x divided by N

5 % 5 = 0

6 % 5 = 1

Homomorphic Encryption (HE)

HE is a special kind of encryption that allows computation on encrypted data

Without decrypting the ciphertext

$$\xrightarrow{\mathrm{Enc}(42)}$$

$$\xleftarrow{\mathrm{Enc}(2 \cdot 42 + 100)}$$

$$N = p \cdot q$$

$$\mathrm{Dec}(\ldots) = 184$$

ECDSA Signature Generation

Ephemeral key

$$k = \mathrm{random}()$$

$$s = \mathrm{sig}(\mathrm{msg}, k, x, \ell)$$

Private key

ECDSA constant

ECDSA signing with 2 parties

Keys

x

k

Key Shares

x_1, x_2

k_1, k_2

Compromising Lindell17 Implementations

- The most popular two-party signing protocol
- Affected: 5 vendors and open-source projects

Lindell17 Key Generation (Step 1/2)

Sample key shards

Chooses a random key share

x_1

Chooses a random key share

x_2

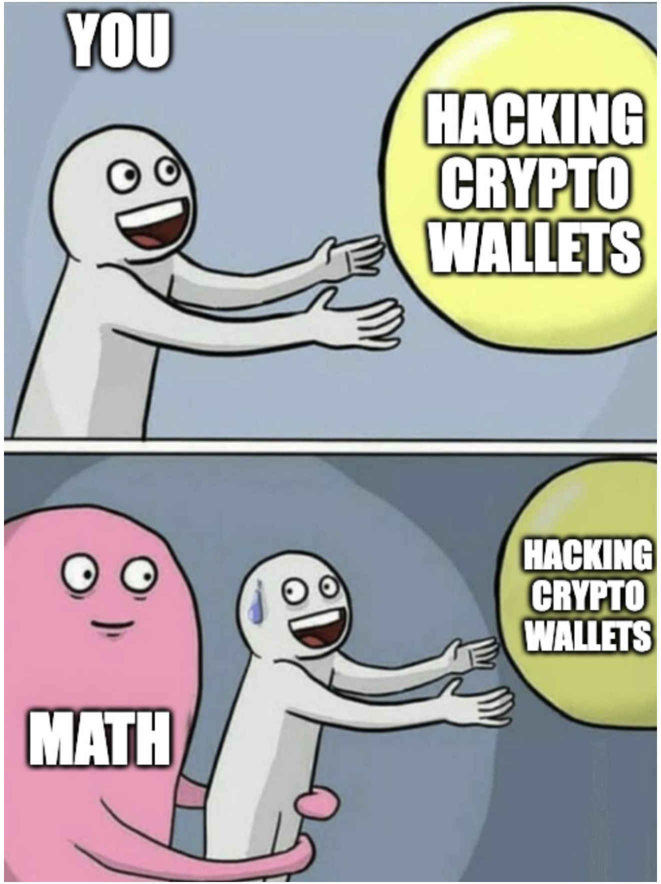
Lindell17 Key Generation (Step 2/2)

Saving Bob's key share under HE

$$\mathrm{Enc}(x_2), N$$

(only bob can can decrypt it,
but alice can operate on it)

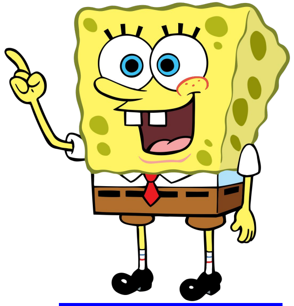
Encrypts their x_2
using their HE key N


YOU
HACKING CRYPTO WALLETS
HACKING CRYPTO WALLETS
MATH

Lindell17 Signing (Step 1/2)

Alice sends a encrypted partial signature

$$\mathrm{Enc}\left(\left(k_1^{-1} \,\%\, \ell\right) \cdot \left(\mathrm{msg}+x_1 \cdot x_2\right)\right)$$



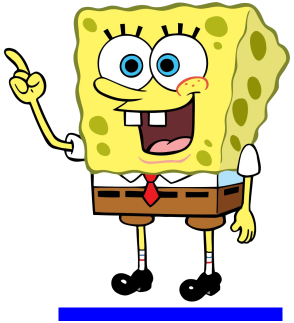
Lindell17 Signing (Step 2/2)

Bob finalizes the signature

$$\text{Decrypt}(\ldots)$$

$$\downarrow$$

$$s = k_2^{-1} \cdot (k_1^{-1} \,\%\, \ell) \cdot (\text{msg} + x_1 \cdot x_2) \,\%\, \ell$$

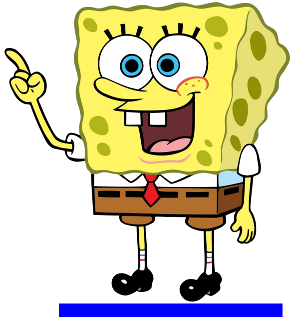


Bob then verifies the signature is valid

What if alice deviates from the protocol?

Hey! the signature is invalid

$$\mathrm{Enc}\left(\left(\cancel{k_1^{-1} \,\%\, \ell}\right)\cdot\left(\mathrm{msg} + x_1 \cdot x_2\right)\right)$$



Bob fails to verify the resulting signature!

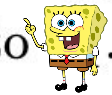
What does the paper say about that?

This trivially implies security when the signing protocol is run sequentially between two parties, since any abort will imply no later executions.

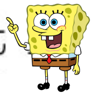
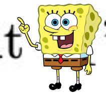
15

Denial-of-Service Attack

Back to the drawing board

The only problem that remains is that  may send an incorrect s' value to .

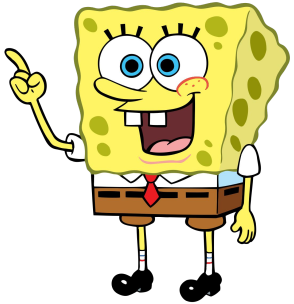
...

In such a case, the mere fact that  aborts or not can leak a single bit about 's private share of the key.

Hypothetical Attack Visualization

s' that fails to finalize if x_2's lsb = 0

Signed successfully



x_2=

0b____

____0

Hypothetical Attack Visualization

s’ that fails to finalize if x_2’s 2nd lsb = 0

Failed to finalize signature

x_2=

0b____
____10

Hypothetical Attack Visualization

s' that fails to finalize if x_2's 3rd lsb = 0

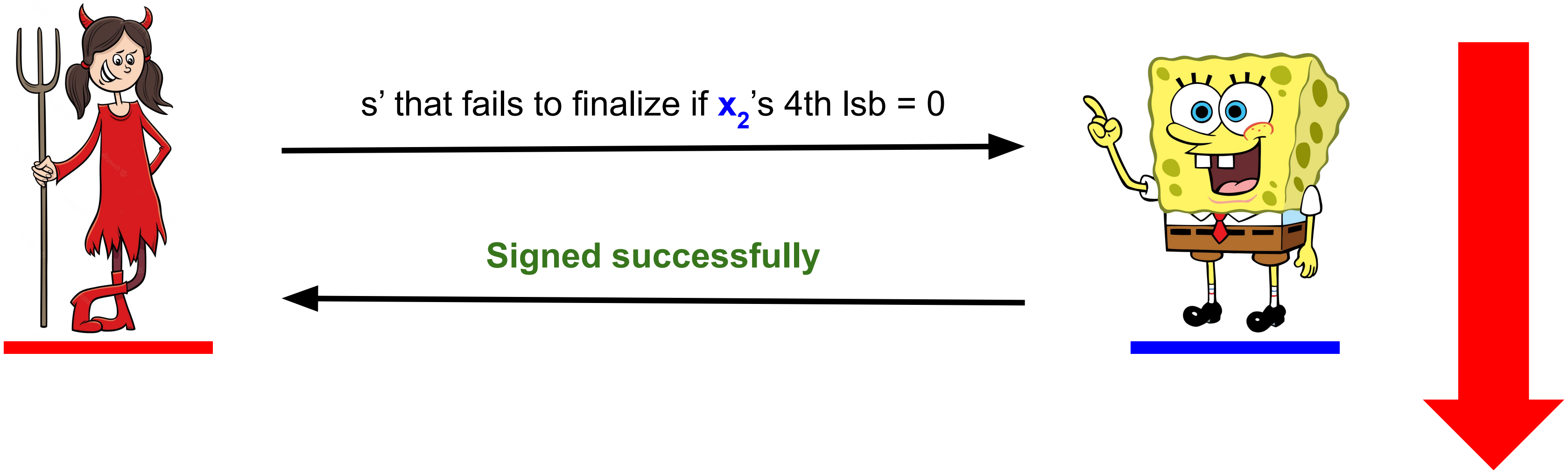
Failed to finalize signature

x_2=

0b____

____110

Hypothetical Attack Visualization



x_2=

```
0b______________________________________________________________________________0110
```

256 signatures later...

Hypothetical Attack Visualization

s’ that fails to finalize if msb is 0

Signed successfully

x_2=

0b0110010111010001011001111111010010101001101010000011001110111001100101101010000010101111100100000101000000000111001001000110000001

01000101101110100011001110001101101010001100101100100010110000010110110010010100111001000100010110001001000001001111011001001100110

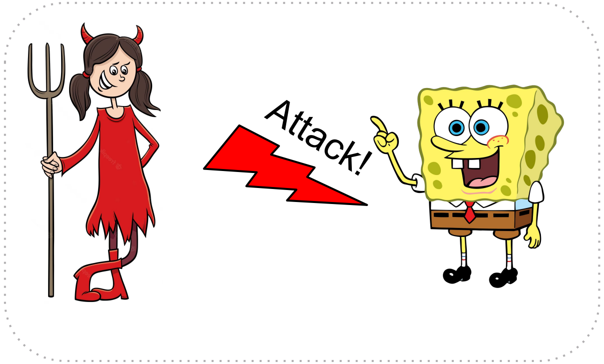
Crafting a malicious partial signature

$$(k_1^{-1} \,\%\, \ell) \cdot (\mathrm{msg} + x_1 \cdot x_2)$$

After  decrypts, **=** iff $x_2 \,\%\, k_1 = 0$

$$(k_1^{-1} \,\underset{\% N}{\cancel{\%\, \ell}}) \cdot (\mathrm{msg} + x_1 \cdot x_2)$$

Obtaining leakage on x2



Signature is valid

$$x_2 \,\%\, k_1 = 0$$

Signature is invalid

$$x_2 \,\%\, k_1 \neq 0$$

Exfiltrating the first bit

$$k_1 = 2$$

Leakage: $x_2 \% 2 = 0$

Exfiltrating the next bit

$$k_1 = 4$$

Leakage: $x_2 \,\%\, 4 = 0$

Wanted: $(x_2 - 1) \,\%\, 4 = 0$

Offsetting previous leaked bits

$$(k_1^{-1} \,\%\, N) \cdot (\mathrm{msg} + x_1 \cdot x_2)$$

$$+$$

$$(k_1^{-1} \,\%\, \ell - k_1^{-1} \,\%\, N) \cdot (\mathrm{msg} + x_1 \cdot \text{known})$$

The previously leaked bits

Exfiltrating the i-th bit

$$k_1 = 2^i$$

Offset: $(k_1^{-1} \% \ell - k_1^{-1} \% N) \cdot (\text{msg} + x_1 \cdot \text{known})$

Leakage: i-th bit

```
./run_poc.sh
```

github.com/ZenGo-X/multi-party-ecdsa

Star 848

How to mitigate the Attack

1. Follow the paper's recommendation (e.g. don't sign again after failure)

Zero-Knowledge Proofs (ZKPs)

Proofs that yield the validity of a statement **and nothing else**

$$\mathrm{Enc}(x)$$

& **ZKP** that $x \in \{1,\ldots,42\}$

Bob verifies the ZKP and is convinced that x is number between 1 and 42

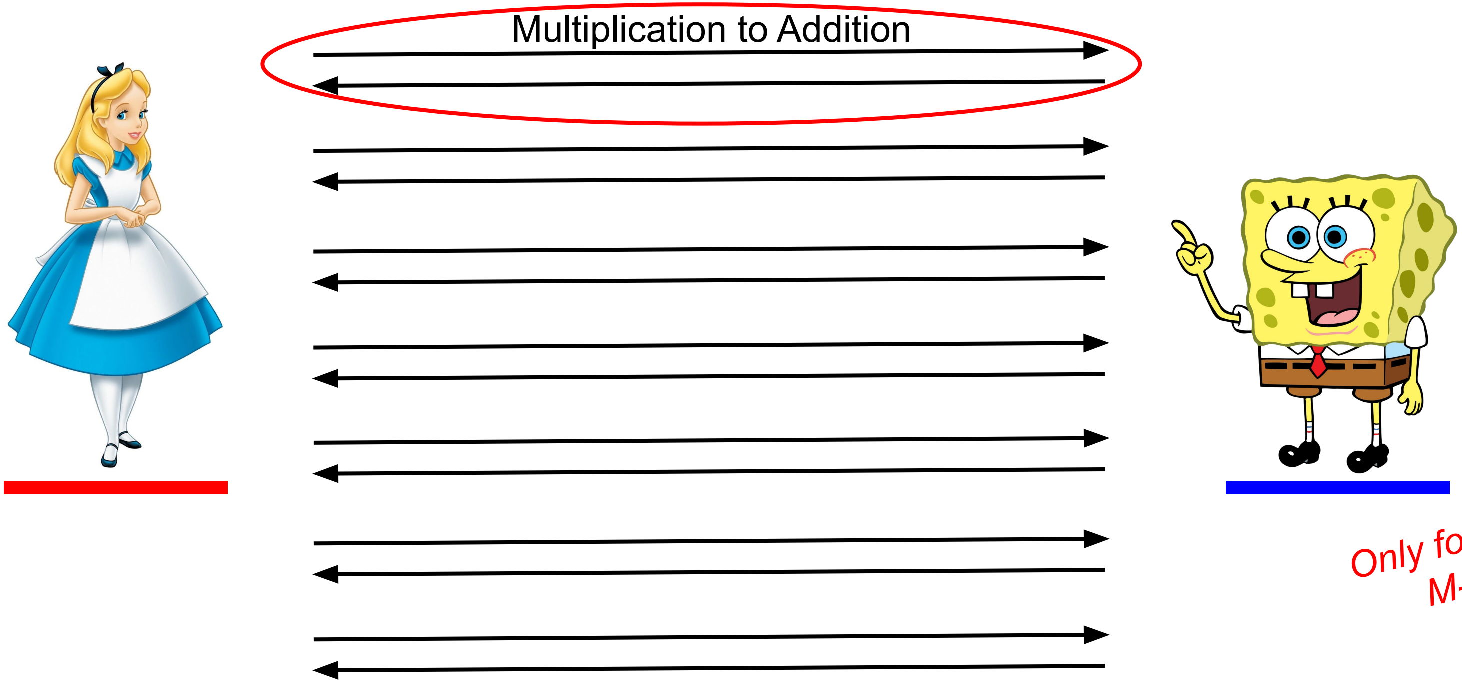
How to mitigate the attack

1. Follow the paper's recommendation (never sign again after failure)
2. Use a ZKP for proving correctness of Alice's message

Compromising GG18 / GG20

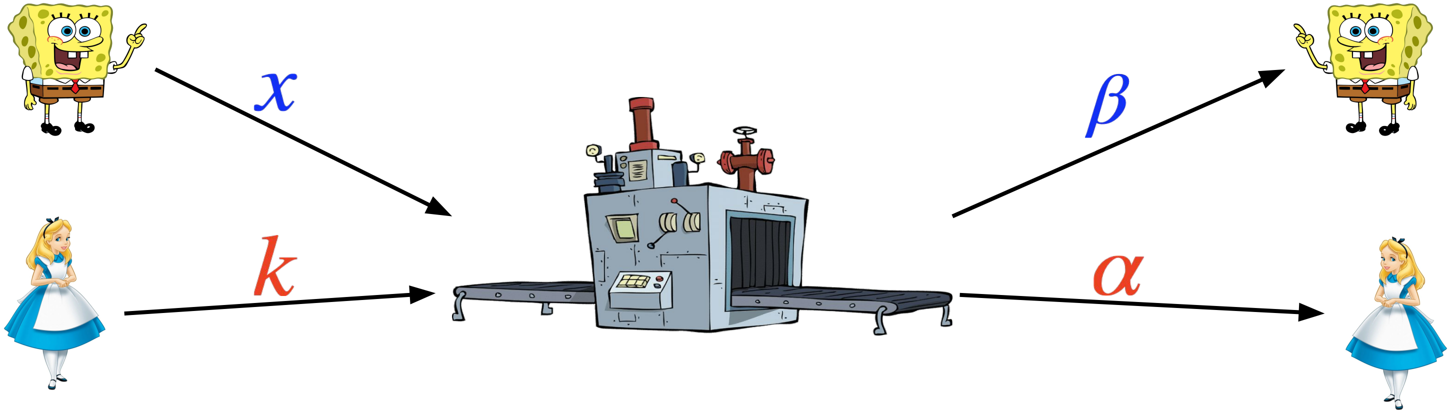
- The most popular multi-party (2+) signing protocol
- Affected: more than 10 vendors and open source projects

The GG protocols are complicated



Only focus on M-t-A

Multiplication to Addition



$$k \cdot x = \alpha + \beta$$

Implementing MtA

$$\mathrm{Enc}(k)\ \&\ \mathrm{ZKP} \longrightarrow$$

$$\longleftarrow \mathrm{Enc}(x \cdot k + \text{mask})\ \&\ \mathrm{ZKP}$$

Alice gets $\alpha = x \cdot k + \text{mask}$

Bob gets $\beta = -\text{mask}$

x & k are 256 bits, and mask is bigger than 512 bits

How does it mask X?

```
x      = 0x1337
mask   = 0x4242424242
k      = 0x6789
x*k    = 0x7c5696f
```

$x \cdot k + \text{mask}$ = 0x424a07abb1

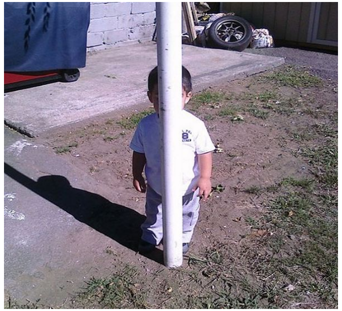
What happens if k > mask?

Key Insight:
Alice has full control over k!

```
x      = 0x1337
mask   = 0x4242424242
k      = 0x10000000000
x*k    = 0x133700000000000
```

$x \cdot k + \text{mask}$ = 0x13370424242424242

The most significant bits leak x



But… the there is a ZK range proof for k

ZK Range Proof

$$(\ldots, z)$$

$$z = w + k \cdot \mathrm{Hash}(w) \,\%\, N$$

Verifier accepts if **…** and ***z* is small**

How to Cheat in the ZKP

We want this value to be “zeroed out”

$$z = w + k \cdot \text{Hash}(w) \,\%\, N$$

Chinese Remainder Theorem (CRT)

If

$$\begin{cases} k \cdot \mathrm{Hash}(w) \,\%\, q = 0 \\ k \cdot \mathrm{Hash}(w) \,\%\, p = 0 \end{cases}$$

Then

$$k \cdot \mathrm{Hash}(w) \,\%\, N = 0$$

$$N = p \cdot q$$

Choose k = q

If

q % q = 0
by definition!

Then

$$\begin{cases} \cancel{q \cdot \mathrm{Hash}(w) \,\%\, q} = 0 \\ \cancel{q \cdot \mathrm{Hash}(w) \,\%\, p} = 0 \end{cases}$$

$$k \cdot \mathrm{Hash}(w) \,\%\, N = 0$$

What if
hash(w) is a
multiple of p

$$N = p \cdot q$$

Brute force w such that hash(w) % p = 0

If

$$\begin{cases} \cancel{q \cdot \mathrm{Hash}(w) \,\%\, q} = 0 \\ \cancel{q \cdot \mathrm{Hash}(w) \,\%\, p} = 0 \end{cases}$$

Then

$$k \cdot \mathrm{Hash}(w) \,\%\, N = 0$$

N is a 2048-bit RSA modulus

Problem: p is too big!!

~~bitsize(N) = 2048~~
~~bitsize(p) = 1024~~
~~bitsize(q) = 1024~~

bitsize(N) = 2048
bitsize(p) = 16
bitsize(q) = 2032

There is no “no small factors” ZKP

- Phase 3 Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Each player P_i proves in ZK that he knows x_i using Schnorr's protocol [46], that N_i is square-free using the proof of Gennaro, Micciancio, and Rabin [32], and that h_1 h_2 generate the same group modulo N_i.

Remember the MtA formula?

$$x \cdot k + \text{mask}$$

$$x \cdot k + \text{mask} \,\%\, N$$

What happens if k ~ N?

```
x      =  0x1234
mask   =  0x4242424242
N      =  0x1000000000000
k      =  0x100000000000
k*x+m  =  0x123404242424242
```

$$x \cdot k + \text{mask} \,\%\, N = \texttt{0x404242424242}$$

The result only partially leaks x

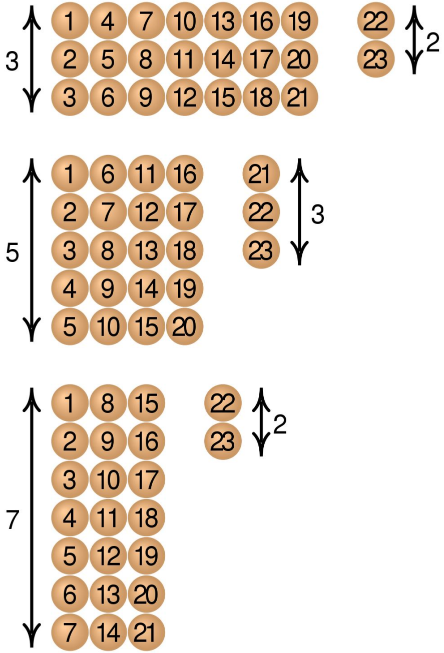
We can obtain a small leakage of x

$$x \% p$$

p is a 16-bit prime

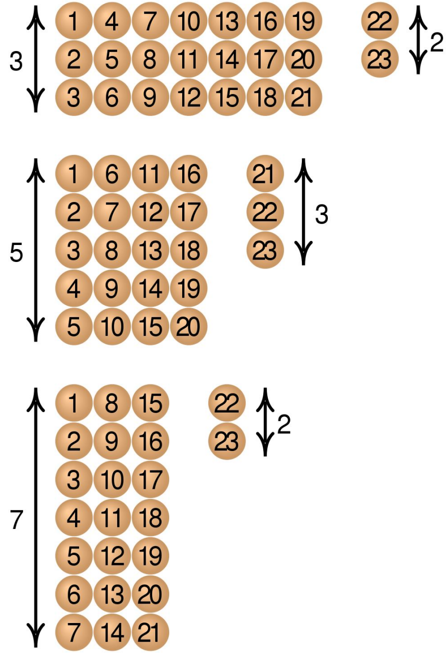
Chinese Remainder Theorem

```
x = 23

x mod 3 = 2
x mod 5 = 3
x mod 7 = 2

CRT((3,2),(5,3),(7,2)) = 23

* It will only work if x is smaller than the
product of the primes (3*5*7=105)
```



Chinese Remainder Theorem

In order to CRT encode a number of size 2^{256}, we need 16 primes of size 2^{16}



So if we can get 16 remainders of x...

$$x \% p_1$$

$$x \% p_2$$

$$\cdots$$

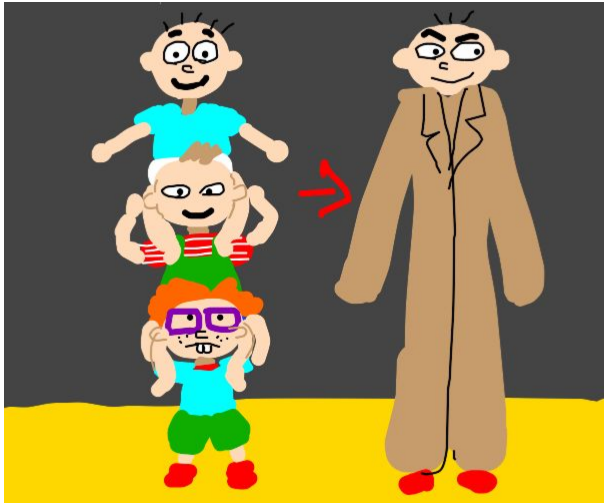
$$x \% p_{16}$$

Problem:
We only have the one N

$$N = p \cdot q$$

What if...

$$N = p \cdot q \qquad\qquad N = p_1 \cdot p_2 \cdot \ldots \cdot p_{16} \cdot q$$



There is no bi-primality ZKP

- Phase 3 Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Each player P_i proves in ZK that he knows x_i using Schnorr's protocol [46], that N_i is square-free using the proof of Gennaro, Micciancio, and Rabin [32], and that h_1 h_2 generate the same group modulo N_i.

How to extract x % Pi

When

$$N = p \cdot q$$

We set

$$k = q$$

To leak

$$x \% p$$

When

$$N = p_1 \cdot p_2 \cdot \ldots \cdot p_{16} \cdot q$$

We set

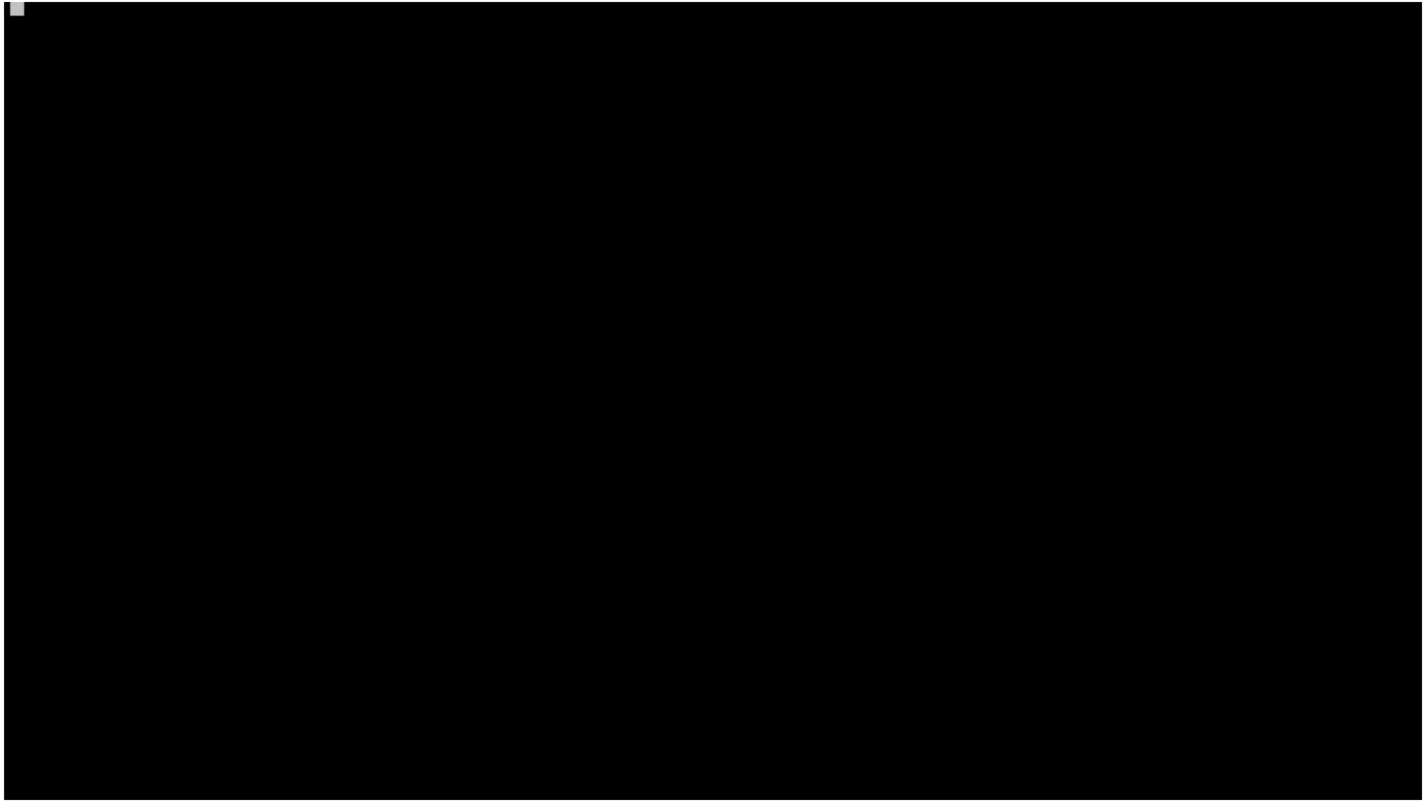
$$k = N/p_i$$

To leak

$$x \% p_i$$

Reconstructing the full key using CRT

```
x mod p1  = 2
x mod p2  = 3
...
x mod p16 = 5

x = CRT((p1,2),(p2,3)...(p16,5))
```



github.com/Safeheron/multi-party-ecdsa-cpp Star 39

How to mitigate the attack

Add ZKPs for proving the well-formedness of Alice's N

Import no small factor proof into GG18/GG20

sword03 committed last month

Compromising the DIY protocol

- Impact: private key exfiltration
- Affected: BitGo TSS
- Published in March 2023

DIY MtA

$$\mathrm{Enc}(k)\ \&\ \cancel{\mathrm{ZKP}}$$

$$\longrightarrow$$

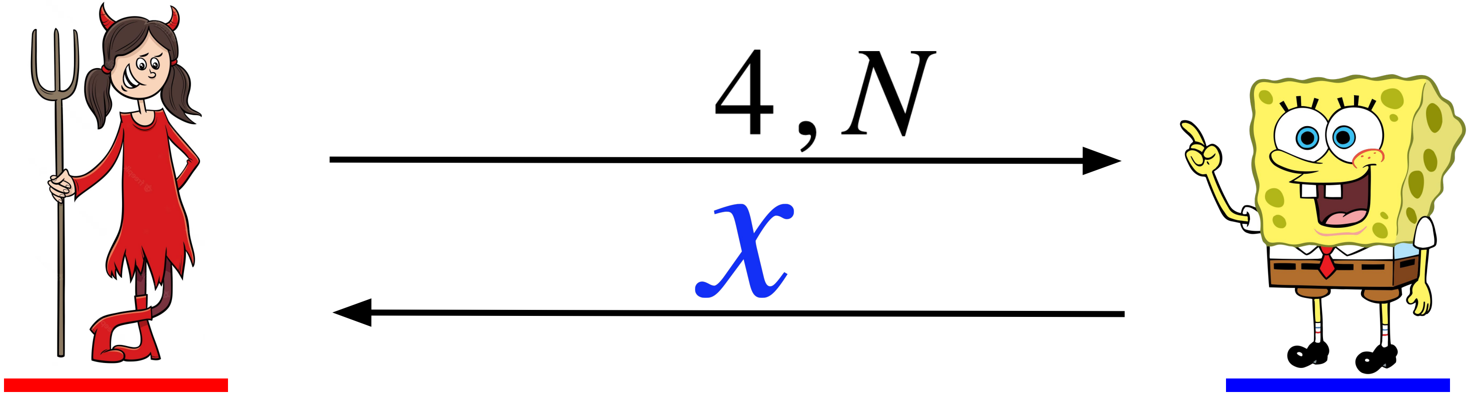
$$\mathrm{Enc}(x \cdot k + \mathsf{mask})\ \&\ \cancel{\mathrm{ZKP}}$$

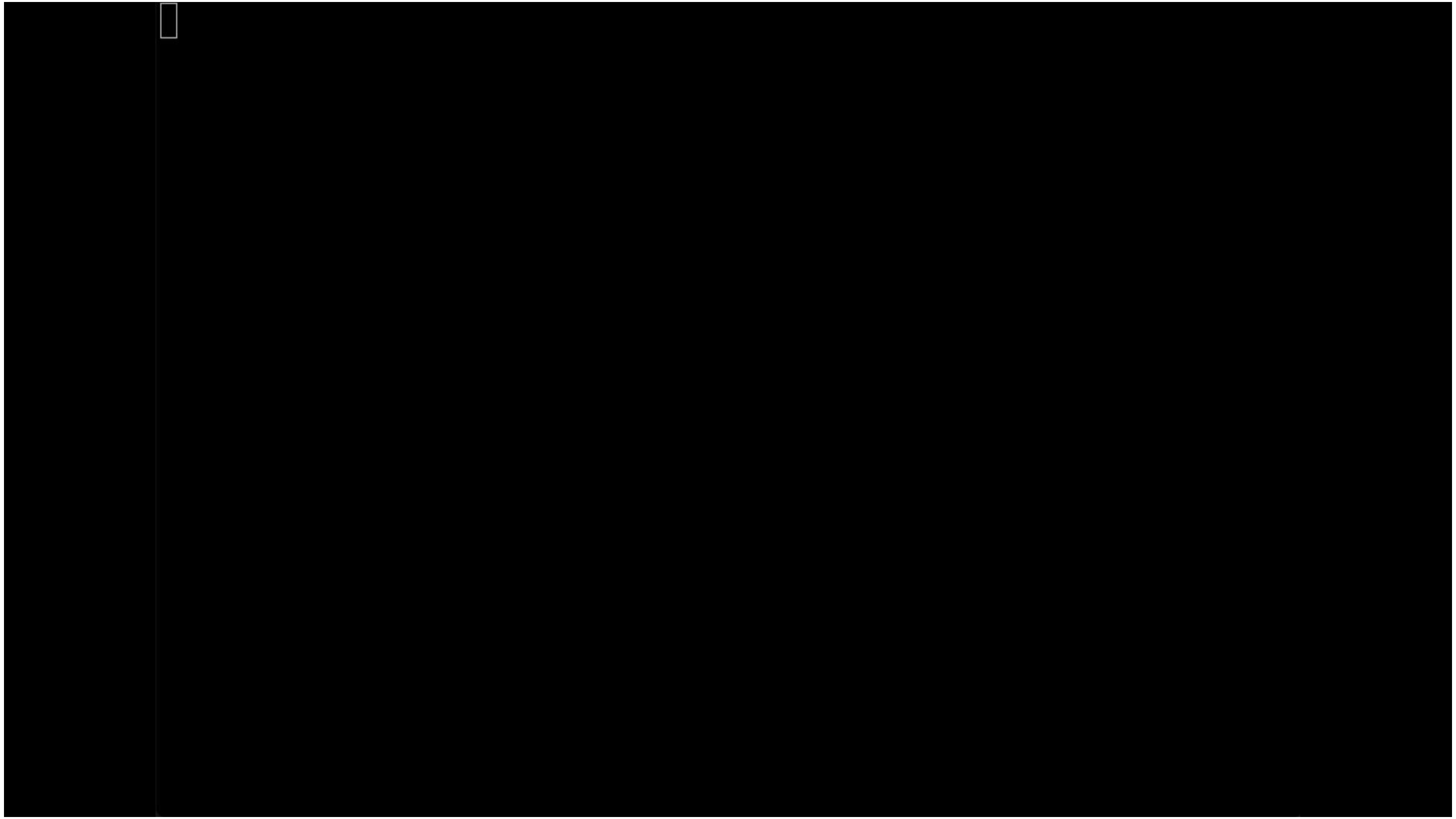
$$\longleftarrow$$

x & k are 256 bits, and **mask is as big as N**

1-signature attack

1. Without the ZKP, Alice can send something that's **not even a ciphertext**
2. By using a maliciously crafted N, Bob will inadvertently send back his x





github.com/BitGo/BitGoJS

Star 323

black hat USA 2023

Concluding Remarks

Black Hat Sound Bytes

- All your keys are belong to us
- MPC is not yet commoditized
- Together we raise the bar for MPC security

Thank you

Proof of concept exploits:

- Lindell17: github.com/fireblocks-labs/zengo-lindell17-exploit-poc
- GG20: github.com/fireblocks-labs/safeheron-gg20-exploit-poc
- DIY: github.com/fireblocks-labs/bitgo-tss-exploit-poc

Technical white paper (for the LaTeX lovers in the crowd):

- github.com/fireblocks-labs/mpc-ecdsa-attacks-23

Follow our research

@nik_mak_

@orenyomtov