Elliptic Curve Cryptography in Practice

Joppe W. Bos

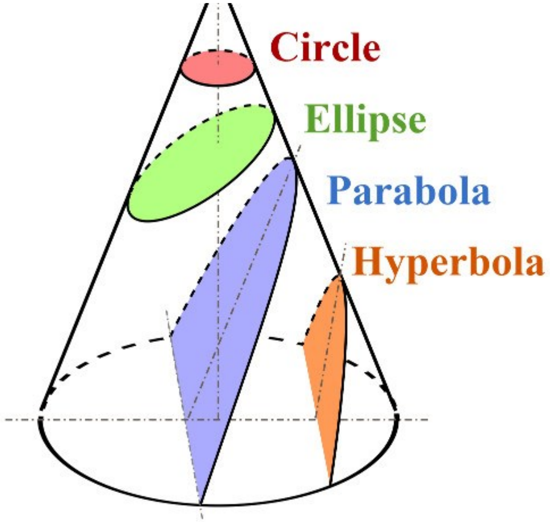
Joint work with
J. Alex Halderman, Nadia Heninger, Jonathan Moore,
Michael Naehrig, Eric Wustrow

Microsoft Research

THE UNIVERSITY OF MICHIGAN · ARTES · SCIENTIA · VERITAS · 1817

LEGES SINE MORIBUS VANAE

Elliptic Curves – An Incomplete Historic Overview

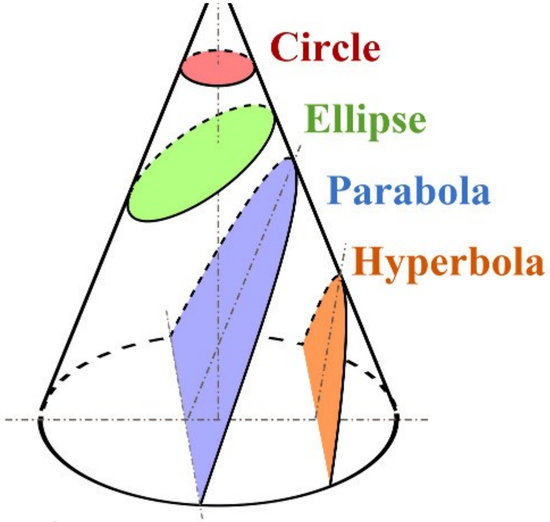


300 BC: Euclid studies conics
262-190 BC: Apollonius of Perga, *On conics*, introducing the name "**ellipse**"
200-300: Diophantus of Alexandria, *Arithmetica*

$$Y(a-Y) = X^3 - X, y = Y - \frac{a}{2}, x = -X \quad \rightarrow \quad y^2 = x^3 - x + (a/2)^2$$

See for more info: Adrian Rice, Ezra Brown: *Why Ellipses Are Not Elliptic Curves* Mathematics Magazine, 2012

Elliptic Curves – An Incomplete Historic Overview



300 BC: Euclid studies conics
262-190 BC: Apollonius of Perga, *On conics*, introducing the name "**ellipse**"
200-300: Diophantus of Alexandria, *Arithmetica*

$$Y(a-Y) = X^3 - X, y = Y - \frac{a}{2}, x = -X \quad \rightarrow \quad y^2 = x^3 - x + (a/2)^2$$

<u>Congruent numbers</u>

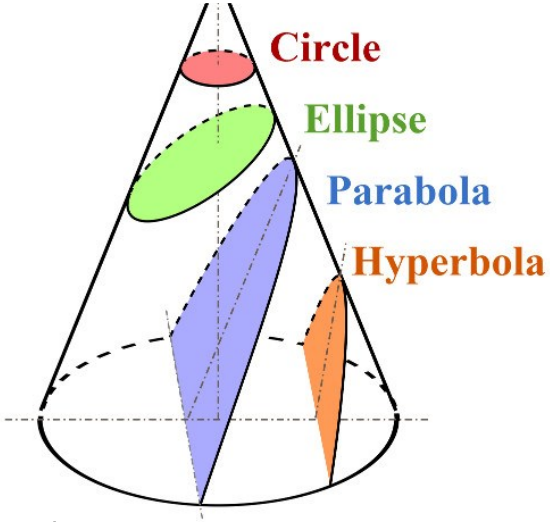
1225: Leonardo of Pisa (Fibonacci), *Liber quadratorum* (The Book of Squares), study of congruent numbers
1621: Claude-Gaspar Bachet de Meziriac translates *Diophantus*
1670: Fermat's notes are published
(problems related to “elliptic curves”)
1730: Euler obtains a copy of Fermat's notes

<u>Ellipse as infinite series</u>

1669: Newton
1733: Euler
1742: Maclaurin

See for more info: Adrian Rice, Ezra Brown: *Why Ellipses Are Not Elliptic Curves* Mathematics Magazine, 2012

Elliptic Curves – An Incomplete Historic Overview



300 BC: Euclid studies conics
262-190 BC: Apollonius of Perga, *On conics*, introducing the name "**ellipse**"
200-300: Diophantus of Alexandria, *Arithmetica*

$$Y(a-Y) = X^3 - X, y = Y - \frac{a}{2}, x = -X \quad \rightarrow \quad y^2 = x^3 - x + (a/2)^2$$

Congruent numbers

1225: Leonardo of Pisa (Fibonacci), *Liber quadratorum* (The Book of Squares), study of congruent numbers
1621: Claude-Gaspar Bachet de Meziriac translates *Diophantus*
1670: Fermat's notes are published
(problems related to "elliptic curves")
1730: Euler obtains a copy of Fermat's notes

Ellipse as infinite series

1669: Newton
1733: Euler
1742: Maclaurin

1825-1828: Legendre, elliptic integrals of the first, second and third kind: elliptic functions
1829: Abel and Jacobi groundbreaking work on elliptic functions
1847: Eisenstein, defines elliptic functions via infinite series and connect elliptic functions with elliptic curves
1863-1864: Clebsch, introduced the idea of using elliptic functions to parameterize cubic curves
Weierstrass, addition formula for elliptic functions to the addition of points on cubic curves
1901: Pointcaré, tied all these ideas together: elliptic curves field as we know it

See for more info: Adrian Rice, Ezra Brown: *Why Ellipses Are Not Elliptic Curves* Mathematics Magazine, 2012

Elliptic Curves in Practice – An Incomplete Overview

1933: Hasse, estimate of the number of points on an elliptic curve

$$|\#E(\mathbf{F}_p) - (p+1)| \leq 2\sqrt{p}$$

1985: Schoof, deterministic polynomial time algorithm for counting points on elliptic curves

1985-1987: Lenstra Jr., elliptic curves can be used to factor integers

Miller & Koblitz, elliptic curves can be used to instantiate *public-key cryptography*

Elliptic Curves in Practice – An Incomplete Overview

1933: Hasse, estimate of the number of points on an elliptic curve

$$|\#E(\mathbf{F}_p) - (p+1)| \leq 2\sqrt{p}$$

1985: Schoof, deterministic polynomial time algorithm for counting points on elliptic curves

1985-1987: Lenstra Jr., elliptic curves can be used to factor integers

Miller & Koblitz, elliptic curves can be used to instantiate *public-key cryptography*

2000: Standard for ECC by Certicom

2006: NIST standard for ECDSA

2006: RFC 4492, ECC in Transport Layer Security (TLS)

2009: RFC 5656, ECC in Secure Shell (SSH)

2009: Nakamoto, Bitcoin

© OpenBSD

Elliptic Curves in Practice – An Incomplete Overview

1933: Hasse, estimate of the number of points on an elliptic curve

$$|\#E(\mathbf{F}_p) - (p+1)| \leq 2\sqrt{p}$$

1985: Schoof, deterministic polynomial time algorithm for counting points on elliptic curves

1985-1987: Lenstra Jr., elliptic curves can be used to factor integers

Miller & Koblitz, elliptic curves can be used to instantiate *public-key cryptography*

2000: Standard for ECC by Certicom

2006: NIST standard for ECDSA

2006: RFC 4492, ECC in Transport Layer Security (TLS)

2009: RFC 5656, ECC in Secure Shell (SSH)

2009: Nakamoto, Bitcoin

OpenSSL Cryptography and SSL/TLS Toolkit

© OpenBSD

bitcoin

2013: **Question #1**

What is the current state of existing elliptic curve deployments in several different applications?

Elliptic Curves in Cryptography - I

$$E : y^2 = x^3 + ax + b$$

- Defined over $\mathbf{F}_p$, where $p > 3$ prime and $a, b \in \mathbf{F}_p$
- Assume $\#E(\mathbf{F}_p) = n$ is prime
- The standard specifies a generator $G \in E(\mathbf{F}_p)$

p	NIST FIPS 186-4	Certicom SEC1	OpenSSL	a
$2^{192} - 2^{64} - 1$	P-192	secp192r1	prime192v1	-3
$2^{224} - 2^{96} + 1$	P-224	secp224r1	secp224r1	-3
$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	P-256	secp256r1	prime256v1	-3
$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$	P-384	secp384r1	secp384r1	-3
$2^{521} - 1$	P-521	secp521r1	secp521r1	-3
$2^{256} - 2^{32} - 977$		secp256k1	secp256k1	0

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^{\times}, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^{\times}, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curve Key Exchange

$(d_a, Q_a), (d_b, Q_b)$ then compute

$P = d_a Q_b = d_b\ Q_a$

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^\times, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curve Key Exchange

$(d_a, Q_a), (d_b, Q_b)$ then compute

$P = d_a Q_b = d_b\ Q_a$

Elliptic Curve Digital Signatures (d, Q, m)

$k \in \mathbf{F}_n^\times$, $\quad kG = (x, y)$, $\quad r = x \bmod n$

$\mathrm{s} = k^{-1}(\mathrm{Hash}(m) + dr) \bmod n$, $\quad$ Signature: (r, s)

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^{\times}, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curve Key Exchange

$(d_a, Q_a), (d_b, Q_b)$ then compute

$P = d_a Q_b = d_b\ Q_a$

Elliptic Curve Digital Signatures (d, Q, m)

$k \in \mathbf{F}_n^{\times}, \quad kG = (x, y), \quad r = x \bmod n$

$\mathrm{s} = k^{-1}(\mathrm{Hash}(m) + dr) \bmod n,$ Signature: (r, s)

We require $r \neq 0 \neq s$ and k is a per-message secret since

if (r, s_1) and (r, s_2) then $k \equiv (s_2 - s_1)^{-1}(e_1 - e_2) \pmod{n}$

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^\times, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curve Key Exchange

$(d_a, Q_a), (d_b, Q_b)$ then compute

$P = d_a Q_b = d_b\ Q_a$

Elliptic Curve Digital Signatures (d, Q, m)

$k \in \mathbf{F}_n^\times, \qquad kG = (x, y), \qquad r = x \bmod n$

$\mathrm{s} = k^{-1}(\mathrm{Hash}(m) + dr) \bmod n,$ Signature: (r, s)

We require $r \neq 0 \neq s$ and k is a per-message secret since

if (r, s_1) and (r, s_2) then $k \equiv (s_2 - s_1)^{-1}(e_1 - e_2) \pmod{n}$

$d \equiv r^{-1}\big(ks - \mathrm{Hash}(m)\big) \pmod{n}$

Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

(d, Q) such that $d \in \mathbf{F}_n^\times, E(\mathbf{F}_p) \ni Q = dG$

Elliptic Curve Key Exchange

$(d_a, Q_a), (d_b, Q_b)$ then compute

$P = d_a Q_b = d_b\ Q_a$

Elliptic Curve Digital Signatures (d, Q, m)

$k \in \mathbf{F}_n^\times, \qquad kG = (x, y), \qquad r = x \bmod n$

$\mathrm{s} = k^{-1}(\mathrm{Hash}(m) + dr) \bmod n, \qquad$ Signature: (r, s)

We require $r \neq 0 \neq s$ and k is a per-message secret since

if (r, s_1) and (r, s_2) then $k \equiv (s_2 - s_1)^{-1}(e_1 - e_2) \pmod n$

$$d \equiv r^{-1}\big(ks - \mathrm{Hash}(m)\big) \pmod n$$

Secp256k1: A *special* curve

secp256k1: $p \equiv 1 \pmod 6$, there exists $\zeta \in \mathbf{F}_p$, such that $\zeta^6 = 1$

$$\psi : E \ \to E, (x, y) \to (\zeta x, -y)$$

Fast scalar multiplication $\psi(P) = \lambda P$ for an integer $\lambda^6 \equiv 1 (\mathrm{mod}\ n)$

R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. CRYPTO 2001

Secure Shell (SSH) Protocol

“The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network.” [RFC4252]

© OpenBSD

Dec. 2009: RFC 5656 “*algorithms based on Elliptic Curve Cryptography (ECC) for use within the Secure Shell (SSH) transport protocol*”

- the ephemeral ECDH key exchange method
- Server (host) authentication (ECDSA)
- Client authentication (ECDSA)

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)

Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

- ✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)

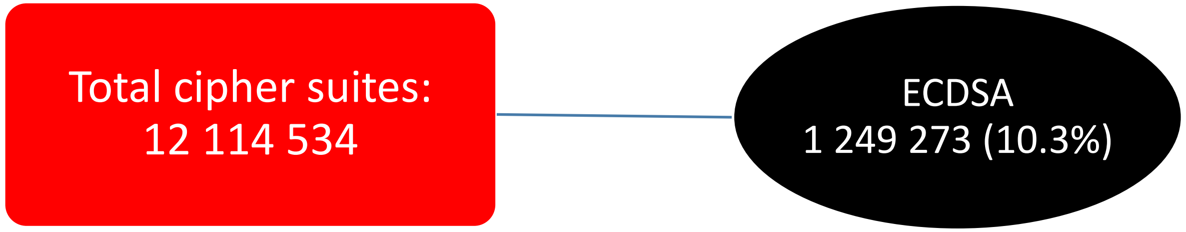
Total cipher suites:
12 114 534

Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)

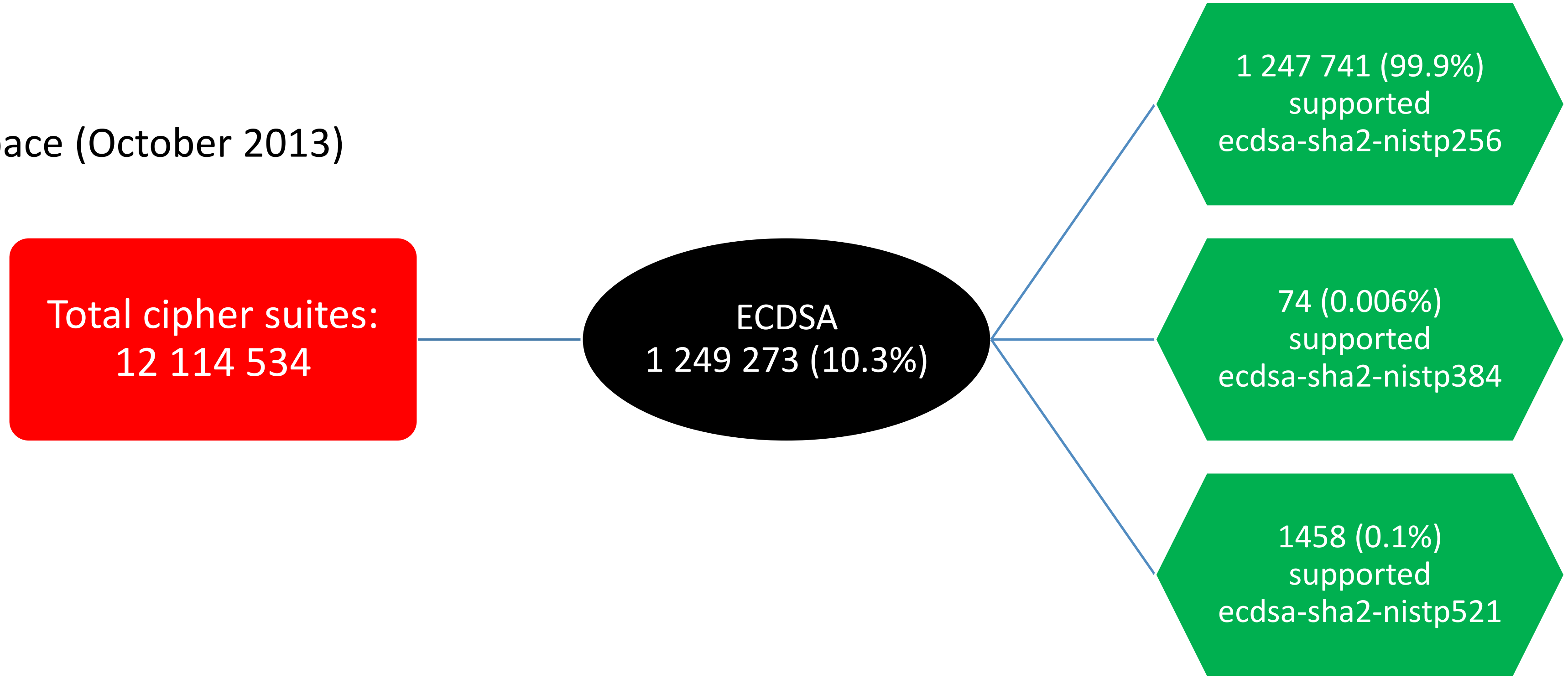


Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

- ✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)

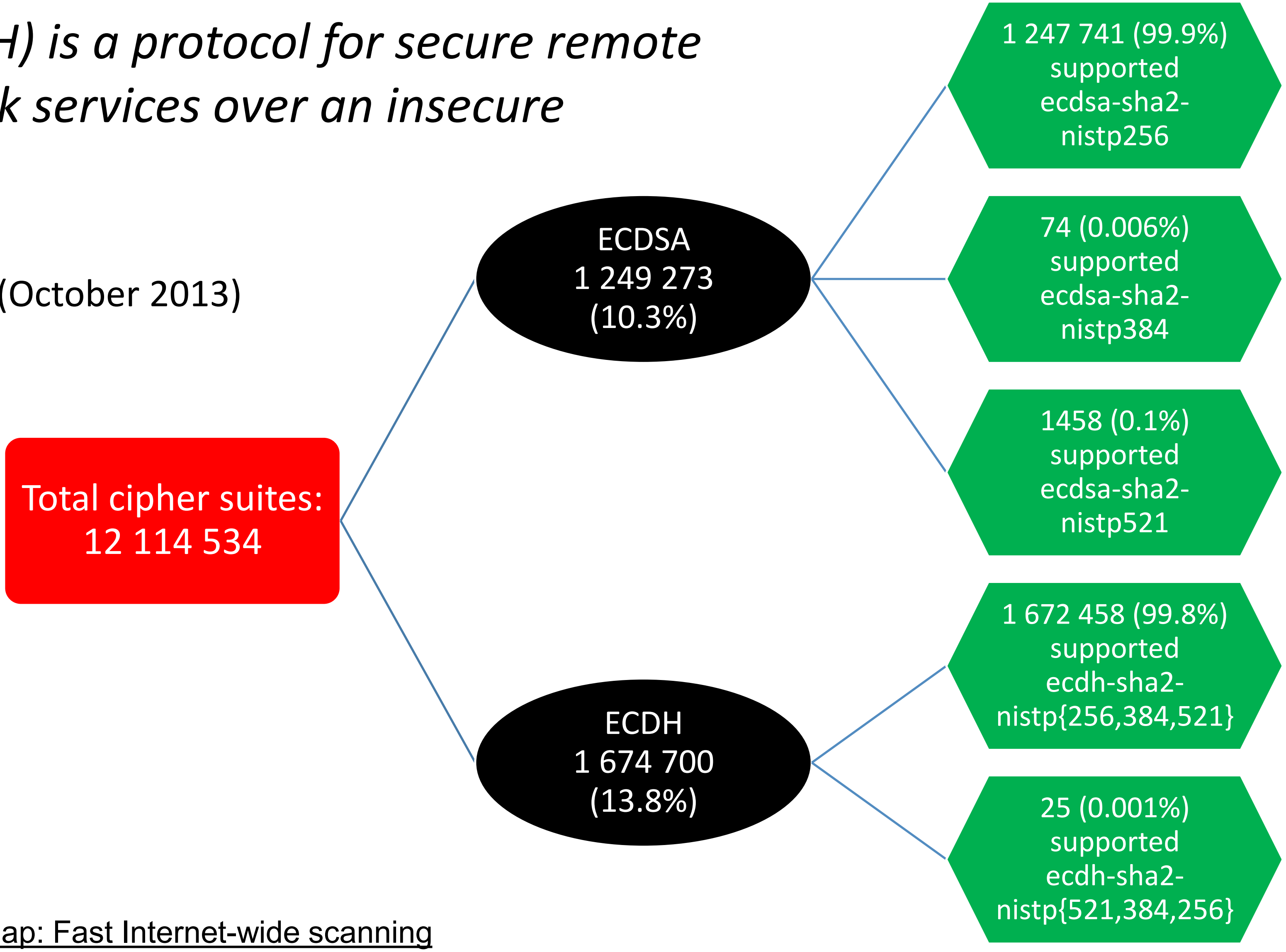


Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

- ✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)

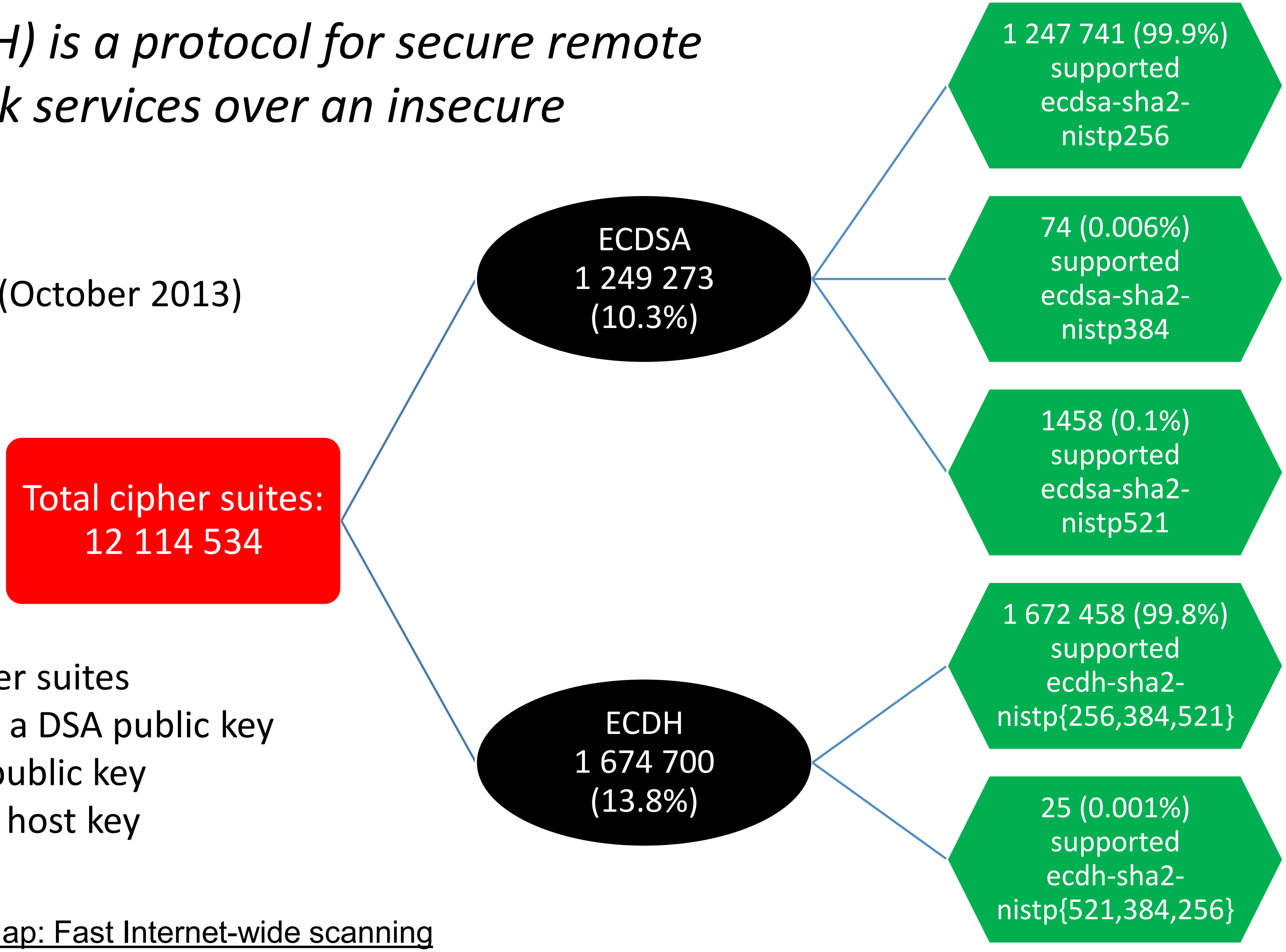


Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Secure Shell (SSH) - Statistics

"The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network." [RFC4252]

- ✓ Scan the complete public IPv4 space (October 2013) for SSH host keys (port 22)



- ✓ Client offered only elliptic curve cipher suites
  - 458 689 servers responded with a DSA public key
  - 29 648 responded with an RSA public key
  - 7 935 responded with an empty host key

Z. Durumeric, E. Wustrow, J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, 2013.

Transport Layer Security (TLS) - Statistics

2006: RFC 4492 “*describes new key exchange algorithms based on Elliptic Curve Cryptography (ECC) for the Transport Layer Security (TLS) protocol. In particular, it specifies the use of Elliptic Curve Diffie-Hellman (ECDH) key agreement in a TLS handshake and the use of Elliptic Curve Digital Signature Algorithm (ECDSA) as a new authentication mechanism.*”

OpenSSL™
Cryptography and SSL/TLS Toolkit

Transport Layer Security (TLS) - Statistics

2006: RFC 4492 "*describes new key exchange algorithms based on Elliptic Curve Cryptography (ECC) for the Transport Layer Security (TLS) protocol. In particular, it specifies the use of Elliptic Curve Diffie-Hellman (ECDH) key agreement in a TLS handshake and the use of Elliptic Curve Digital Signature Algorithm (ECDSA) as a new authentication mechanism.*"

OpenSSL
Cryptography and SSL/TLS Toolkit

Elliptic curve Diffie-Hellman (ECDH) key exchange

- **Long-term**: key is reused for different key exchanges
- **Ephemeral**: key is regenerated for each key exchange

Elliptic curve digital signature (ECDSA)

TLS certificates contain a public key for authentication: either **ECDSA** or **RSA**

Transport Layer Security (TLS) - Statistics

2006: RFC 4492 "*describes new key exchange algorithms based on Elliptic Curve Cryptography (ECC) for the Transport Layer Security (TLS) protocol. In particular, it specifies the use of Elliptic Curve Diffie-Hellman (ECDH) key agreement in a TLS handshake and the use of Elliptic Curve Digital Signature Algorithm (ECDSA) as a new authentication mechanism.*"

OpenSSL
Cryptography and SSL/TLS Toolkit

Elliptic curve Diffie-Hellman (ECDH) key exchange

- **Long-term**: key is reused for different key exchanges
- **Ephemeral**: key is regenerated for each key exchange

Elliptic curve digital signature (ECDSA)

TLS certificates contain a public key for authentication: either **ECDSA** or **RSA**

Example

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

- ephemeral ECDH for a key exchange
- signed with an RSA key for identity verification
- AES-128 in CBC mode for encryption
- SHA-1 in an HMAC for message authentication

Transport Layer Security (TLS) - Statistics

October 2013: scan IPv4 address space (port 443)

- TLS server **does not** send its full list of cipher suites it supports
- Client sends its list, server picks a single cipher suite or closes connection

Transport Layer Security (TLS) - Statistics

Idea:

$L =$ a set of 38 ECDH and ECDHE cipher suites (28 different curves)

repeat {

  connect to server with L

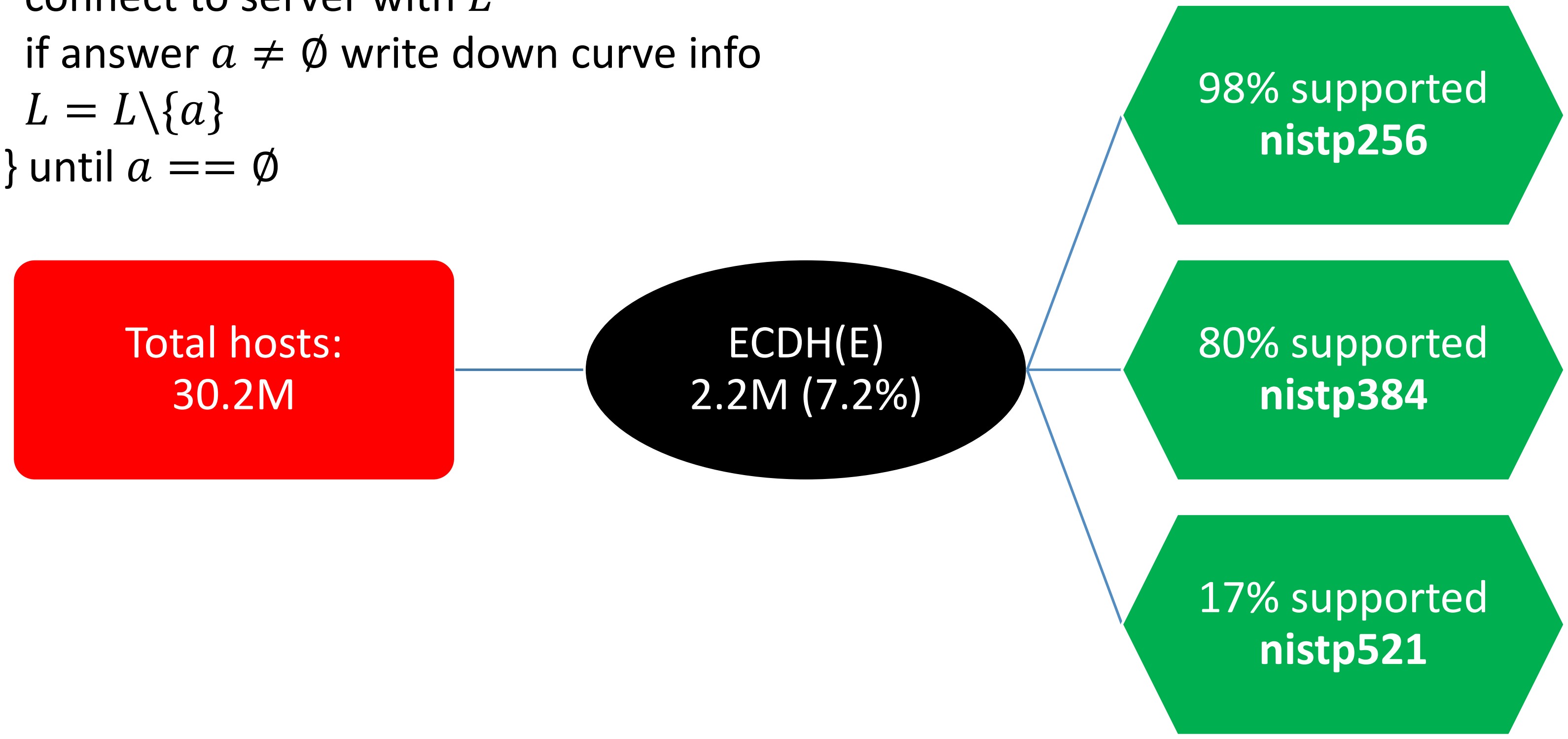
  if answer $a \neq \emptyset$ write down curve info

  $L = L \setminus \{a\}$

} until $a == \emptyset$

Transport Layer Security (TLS) - Statistics

Idea:
L = a set of 38 ECDH and ECDHE cipher suites (28 different curves)
repeat {
  connect to server with L
  if answer $a \neq \emptyset$ write down curve info
  $L = L \setminus \{a\}$
} until $a == \emptyset$



Transport Layer Security (TLS) - Statistics

Idea:

L = a set of 38 ECDH and ECDHE cipher suites (28 different curves)

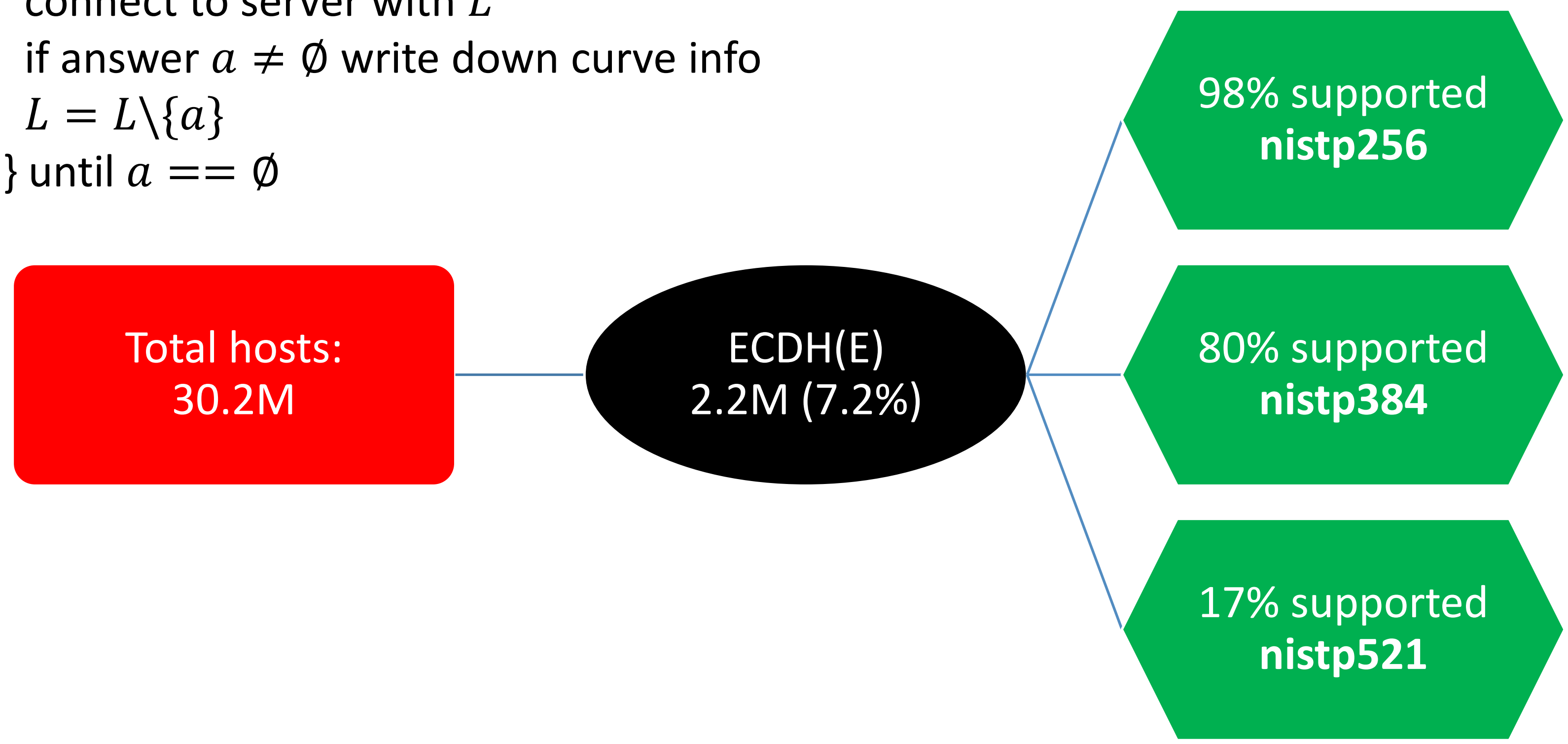
repeat {

connect to server with L

if answer $a \neq \emptyset$ write down curve info

$L = L \backslash \{a\}$

} until $a == \emptyset$



1.7 million hosts supported > 1 curve

98.9% support a strictly increasing curve-size preference.

354 767 hosts

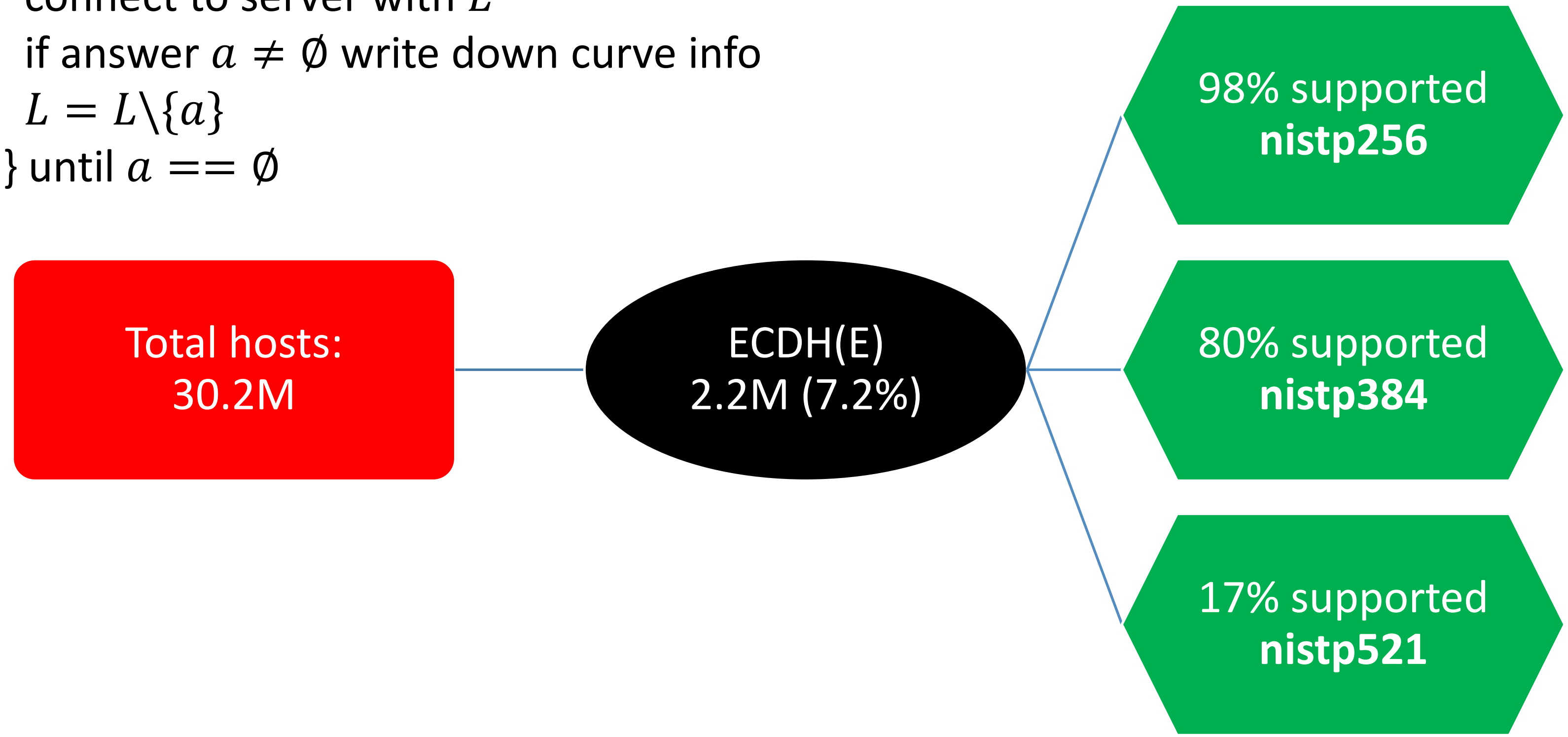
"secp256r1, secp384r1, secp521r1"

190 hosts

"secp521r1, secp384r1, secp256r1"

Transport Layer Security (TLS) - Statistics

Idea:
$L =$ a set of 38 ECDH and ECDHE cipher suites (28 different curves)
repeat {
  connect to server with L
  if answer $a \neq \emptyset$ write down curve info
  $L = L \setminus \{a\}$
} until $a == \emptyset$



1.7 million hosts supported > 1 curve
98.9% support a strictly increasing curve-size preference.

354 767 hosts
“secp256r1, secp384r1, secp521r1”
190 hosts
“secp521r1, secp384r1, secp256r1"

Hosts prefer lower computation and bandwidth costs over increased security

Bitcoin - Statistics

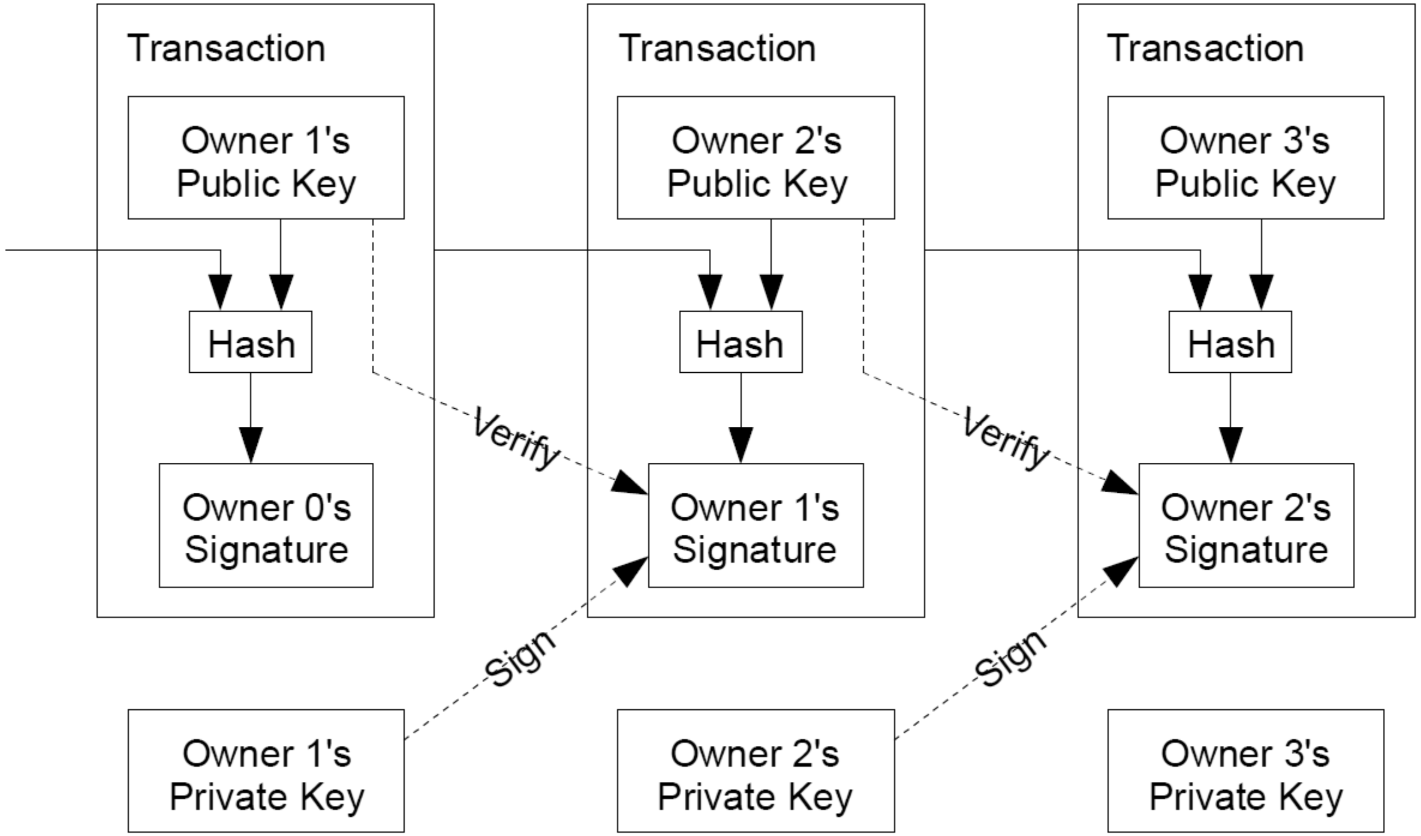
Bitcoin is a distributed peer-to-peer digital currency which allows “*online payments to be sent directly from one party to another without going through a financial institution*"

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

Bitcoin - Statistics

Bitcoin is a distributed peer-to-peer digital currency which allows “*online payments to be sent directly from one party to another without going through a financial institution*"

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.



- All transactions are public
- From asymmetric crypto point of view Bitcoin relies exclusively on ECDSA
- Interesting choice:
  not NIST P-256 but “special” sec256k1
- Avoiding double spending etc. is out of scope for this talk

Bitcoin - Statistics

Bitcoin is a distributed peer-to-peer digital currency which allows *"online payments to be sent directly from one party to another without going through a financial institution*"

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

Bitcoin address is not really an ECDSA key *K*

HASH160 = RIPEMD160(SHA256(*K*))

Bitcoin address = $\mathrm{base58}\left(0x00 \parallel \mathrm{HASH160} \parallel \left\lfloor \frac{\mathrm{SHA256}(\mathrm{SHA256}(0x00 \parallel \mathrm{HASH160}))}{2^{224}} \right\rfloor\right)$

Bitcoin - Statistics

Bitcoin is a distributed peer-to-peer digital currency which allows *"online payments to be sent directly from one party to another without going through a financial institution*"

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

Bitcoin address is not really an ECDSA key *K*

HASH160 = RIPEMD160(SHA256(*K*))

$$\text{Bitcoin address} = \text{base58}\left(0\text{x}00 \parallel \text{HASH160} \parallel \left\lfloor \frac{\text{SHA256}(\text{SHA256}(0\text{x}00 \parallel \text{HASH160}))}{2^{224}} \right\rfloor\right)$$

August 2013: Bitcoin block chain (#252 450)

- Extracted **22M** transactions (26GB plaintext file)
- **46M** signatures
- **46M** ECDSA keys
  - **15.3M** unique
  - **6.6M** compressed
  - **8.7M** uncompressed
  - (**136** points both)

Bitcoin - Statistics

Bitcoin is a distributed peer-to-peer digital currency which allows “*online payments to be sent directly from one party to another without going through a financial institution*"

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

Bitcoin address is not really an ECDSA key *K*

HASH160 = RIPEMD160(SHA256(*K*))

Bitcoin address = $\mathrm{base58}\left(0\mathrm{x}00 \parallel \mathrm{HASH160} \parallel \left\lfloor \frac{\mathrm{SHA256}(\mathrm{SHA256}(0\mathrm{x}00 \| \mathrm{HASH160}))}{2^{224}} \right\rfloor\right)$

August 2013: Bitcoin block chain (#252 450)

- Extracted **22M** transactions (26GB plaintext file)
- **46M** signatures
- **46M** ECDSA keys
  - **15.3M** unique
  - **6.6M** compressed
  - **8.7M** uncompressed
  - (**136** points both)

October 2013: **> 11.5 million** bitcoins in circulation
estimated value: **> 2 billion** USD

Public Key Cryptography in Practice

A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, C. Wachter: Public Keys, in *Crypto 2012*

- Millions of keys from TLS X.509 certs (EFF SSL Observatory)
- Millions of PGP keys

"two out of every one thousand RSA moduli collected offer no security"

N. Heninger, Z. Durumeric, E. Wustrow, J. A. Halderman: Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices, in *USENIX Security Symposium 2012*

- Millions of keys from TLS X.509 certs (scan IPv4 network)
- Millions of PGP keys

"we are able to obtain the private keys for 0.50% of TLS hosts and 0.03% of SSH hosts"

Likely cause: limited entropy in (embedded) devices

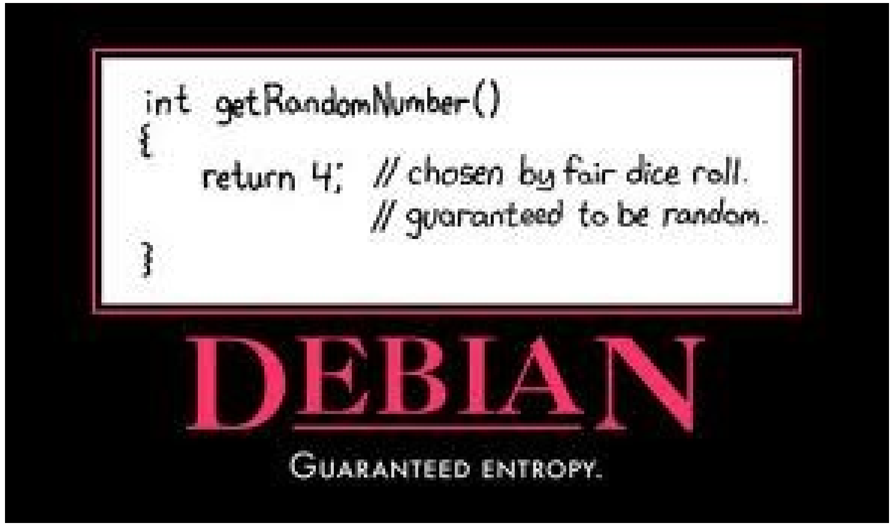
Cryptographic Mistakes / Malfunctions in Practice

2008: Debian OpenSSL vulnerability
change in the code (2006) prevented any entropy from being incorporated into the OpenSSL entropy pool

2012: RSA keys with common factors (previous slide)

2013: RSA keys obtained from Taiwan's national Citizen Digital Certificate database can be factored due to a malfunctioning hardware random number generator on cryptographic smart cards

D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. van Someren: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *ASIACRYPT 2013*.



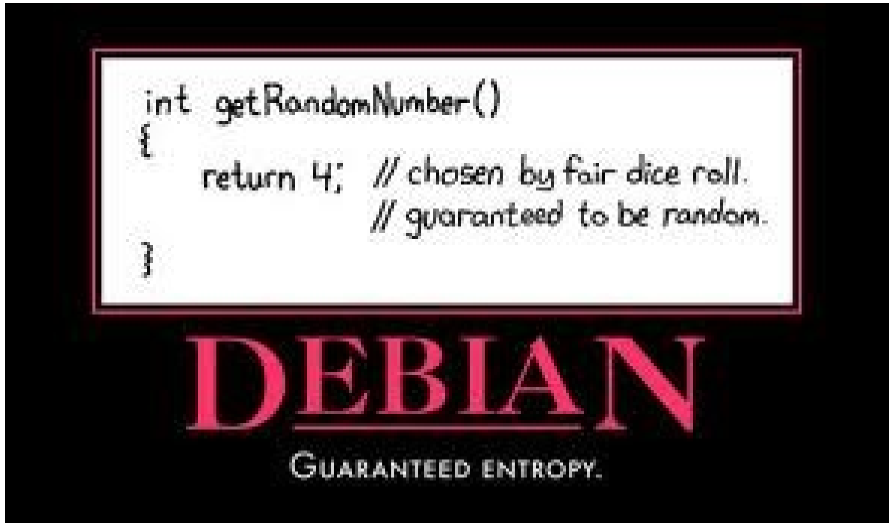
Cryptographic Mistakes / Malfunctions in Practice

2008: Debian OpenSSL vulnerability
change in the code (2006) prevented any entropy from being incorporated into the OpenSSL entropy pool

2012: RSA keys with common factors (previous slide)

2013: RSA keys obtained from Taiwan's national Citizen Digital Certificate database can be factored due to a malfunctioning hardware random number generator on cryptographic smart cards
D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. van Someren: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *ASIACRYPT 2013*.

2010: Sony PlayStation 3 video game console
The PS3 used a constant value for ECDSA signatures allowing hackers to compute the secret code signing key
“Bushing", H. M. Cantero, S. Boessenkool, and S. Peter. PS3 epic fail, *27th Chaos Communication Congress*



Cryptographic Mistakes / Malfunctions in Practice

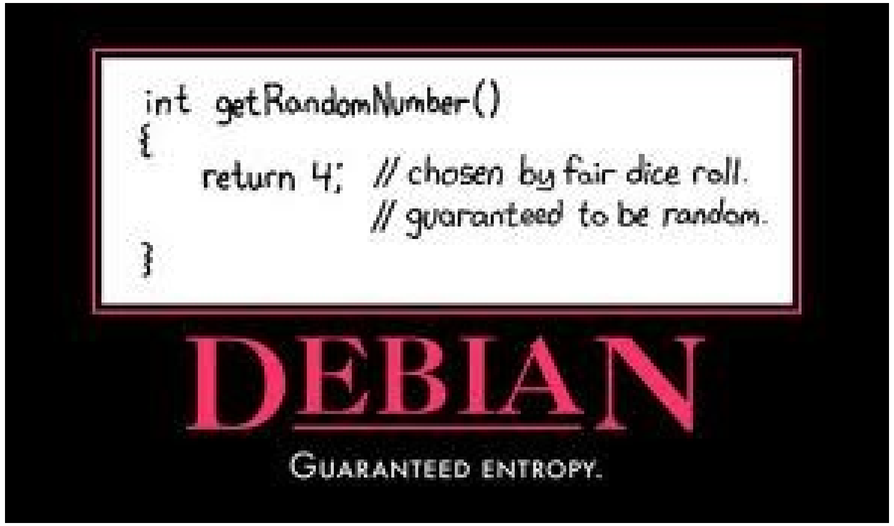
2008: Debian OpenSSL vulnerability
change in the code (2006) prevented any entropy from being incorporated into the OpenSSL entropy pool

2012: RSA keys with common factors (previous slide)

2013: RSA keys obtained from Taiwan's national Citizen Digital Certificate database can be factored due to a malfunctioning hardware random number generator on cryptographic smart cards.
D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. van Someren: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *ASIACRYPT 2013*.

2013: **Question #2**

2010: Sony PlayStation 3 video game console
The PS3 used a constant value for ECDSA signatures allowing hackers to compute the secret code signing key
"Bushing", H. M. Cantero, S. Boessenkool, and S. Peter. PS3 epic fail, *27th Chaos Communication Congress*



Can we find problems that might signal the presence of cryptographic vulnerabilities in ECC?

Cryptographic “Sanity” Checks

Key Generation

$Q = dG$ poor randomness might result in repeated d

RSA	$p_1 q_1$
$p_1 q_1$	?
$p_1 q_2$	Insecure
$p_2 q_1$	Insecure
$p_2 q_2$	Secure

ECC	d_1
d_1	?
d_2	Secure

Cryptographic “Sanity” Checks

Key Generation

$Q = dG$ poor randomness might result in repeated d

RSA	$p_1 q_1$
$p_1 q_1$	?
$p_1 q_2$	Insecure
$p_2 q_1$	Insecure
$p_2 q_2$	Secure

ECC	d_1
d_1	?
d_2	Secure

Repeated Per-Message Signature Secrets

Signature (r_i, s_i) if we find (for $i \neq j$)

$$(r_i, s_i), (r_i, s_j)$$

Then we can compute the secret key

Cryptographic "Sanity" Checks

Key Generation

$Q = dG$ poor randomness might result in repeated d

RSA	$p_1 q_1$
$p_1 q_1$	?
$p_1 q_2$	Insecure
$p_2 q_1$	Insecure
$p_2 q_2$	Secure

ECC	d_1
d_1	?
d_2	Secure

Repeated Per-Message Signature Secrets

Signature (r_i, s_i) if we find (for $i \neq j$)

$$(r_i, s_i), (r_i, s_j)$$

Then we can compute the secret key

Unexpected, Illegal, and Known Weak Values

Generate many scalars s and check if sG occurs in practice for NISTp256 and secp256k1

- *Negation*: store x-coordinate only (we represent both $\pm sG$)
- *Small integers*: $10^0 \leq s \leq 10^6$
- *Bitcoin*: Also $i\lambda G = \psi(iG)$
- *Low Hamming weight*: $\binom{256}{1} = 256,\ \binom{256}{2} = 32640,\ \binom{256}{3} = 2763520$

SSH/TLS - Cryptographic Sanity Checks

	SSH	TLS
# elliptic curve public keys	1.2 million	5.4 million
# unique keys	0.8 million	5.2 million

SSH/TLS - Cryptographic Sanity Checks

	SSH	TLS
# elliptic curve public keys	1.2 million	5.4 million
# unique keys	0.8 million	5.2 million

SSH

Most commonly repeated keys are from cloud hosting providers

- shared SSH infrastructure that is accessible via multiple IP address
- mistake during virtual machine deployment
  Example:
  July 2013: Digital Ocean, Avoid duplicate SSH host keys
  “The SSH host keys for some Ubuntu-based systems could have been duplicated by DigitalOcean's snapshot and creation process.”

  5614 hosts served the public key from Digital Ocean’s setup guide

SSH/TLS - Cryptographic Sanity Checks

	SSH	TLS
# elliptic curve public keys	1.2 million	5.4 million
# unique keys	0.8 million	5.2 million

SSH

Most commonly repeated keys are from cloud hosting providers

- shared SSH infrastructure that is accessible via multiple IP address
- mistake during virtual machine deployment
  Example:
  July 2013: Digital Ocean, Avoid duplicate SSH host keys
  "The SSH host keys for some Ubuntu-based systems could have been duplicated by DigitalOcean's snapshot and creation process."

  5614 hosts served the public key from Digital Ocean's setup guide

TLS

Many duplicated keys are from small set of subnets, most likely nothing wrong: single shared host, but

- A single key presented by 2000 hosts
- 1800 of a particular brand of devices presented the same NISTp256 key for ECDHE key exchange
  buying this device allows to decrypt traffic

No overlap between SSH and TLS keys

Bitcoin - Cryptographic Sanity Checks

- We collected **47 093 121** elliptic curve points from the signatures and verified that they are correct i.e. the points are on the curve *secp256k1*
- We looked for duplicated nonces in the signatures
  158 unique public keys had used the same signature nonce *r* value in more than one signature
  → making it possible to compute these users' private keys
- Currently only 0.00031217 BTC = 0.1228 USD left on these accounts

Bitcoin - Cryptographic Sanity Checks

- We collected **47 093 121** elliptic curve points from the signatures and verified that they are correct i.e. the points are on the curve *secp256k1*
- We looked for duplicated nonces in the signatures
  158 unique public keys had used the same signature nonce *r* value in more than one signature
  → making it possible to compute these users' private keys
- Currently only 0.00031217 BTC = 0.1228 USD left on these accounts

Address: 1HKywxiL4JziqXrzLKhmB6a74ma6kxbSDj

March to October 2013: 59 BTC ≈ 23000 USD has been stolen from 10 of these addresses

Bitcoin - Cryptographic Sanity Checks

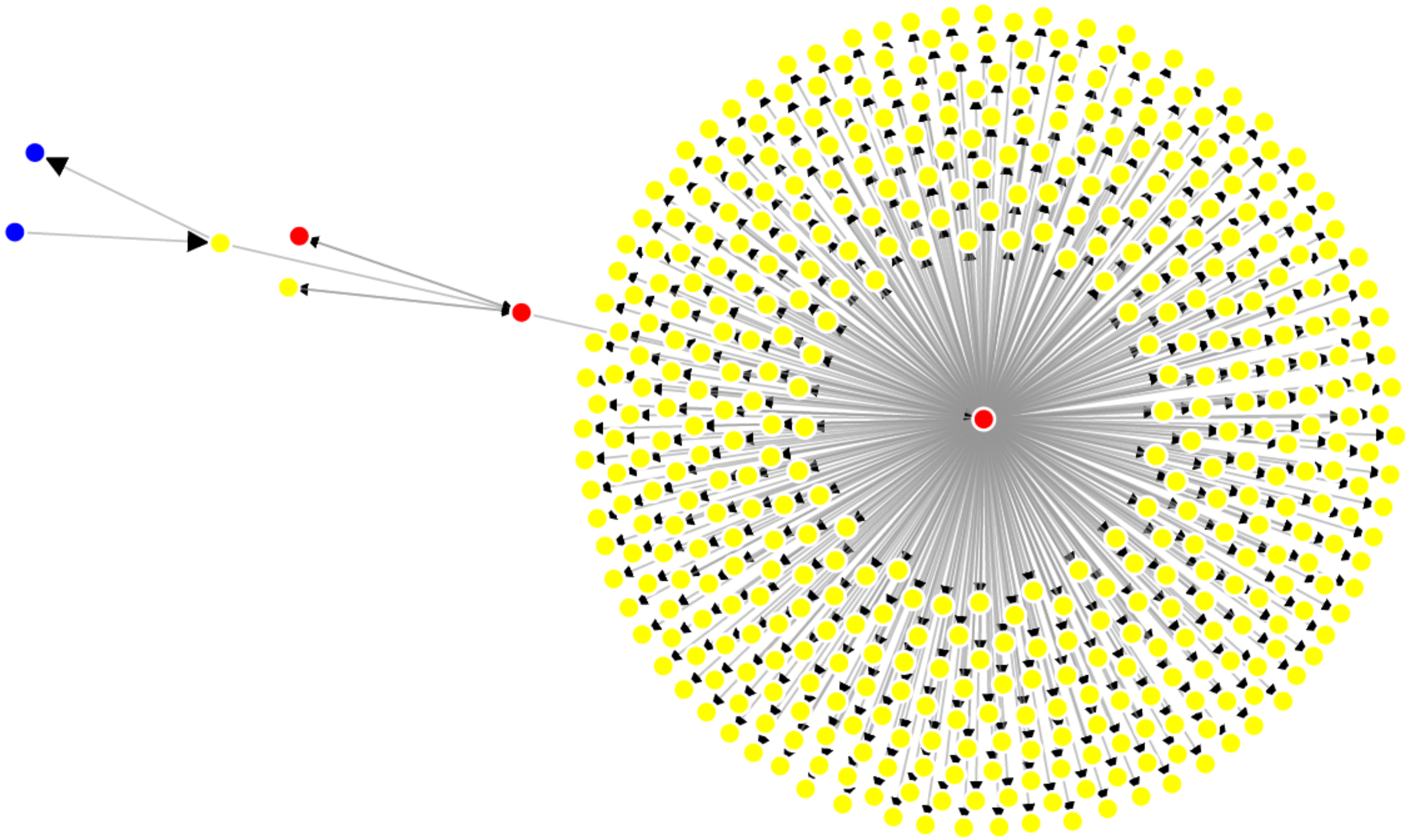
- We collected **47 093 121** elliptic curve points from the signatures and verified that they are correct i.e. the points are on the curve *secp256k1*
- We looked for duplicated nonces in the signatures
  158 unique public keys had used the same signature nonce *r* value in more than one signature
  → making it possible to compute these users' private keys
- Currently only 0.00031217 BTC = 0.1228 USD left on these accounts

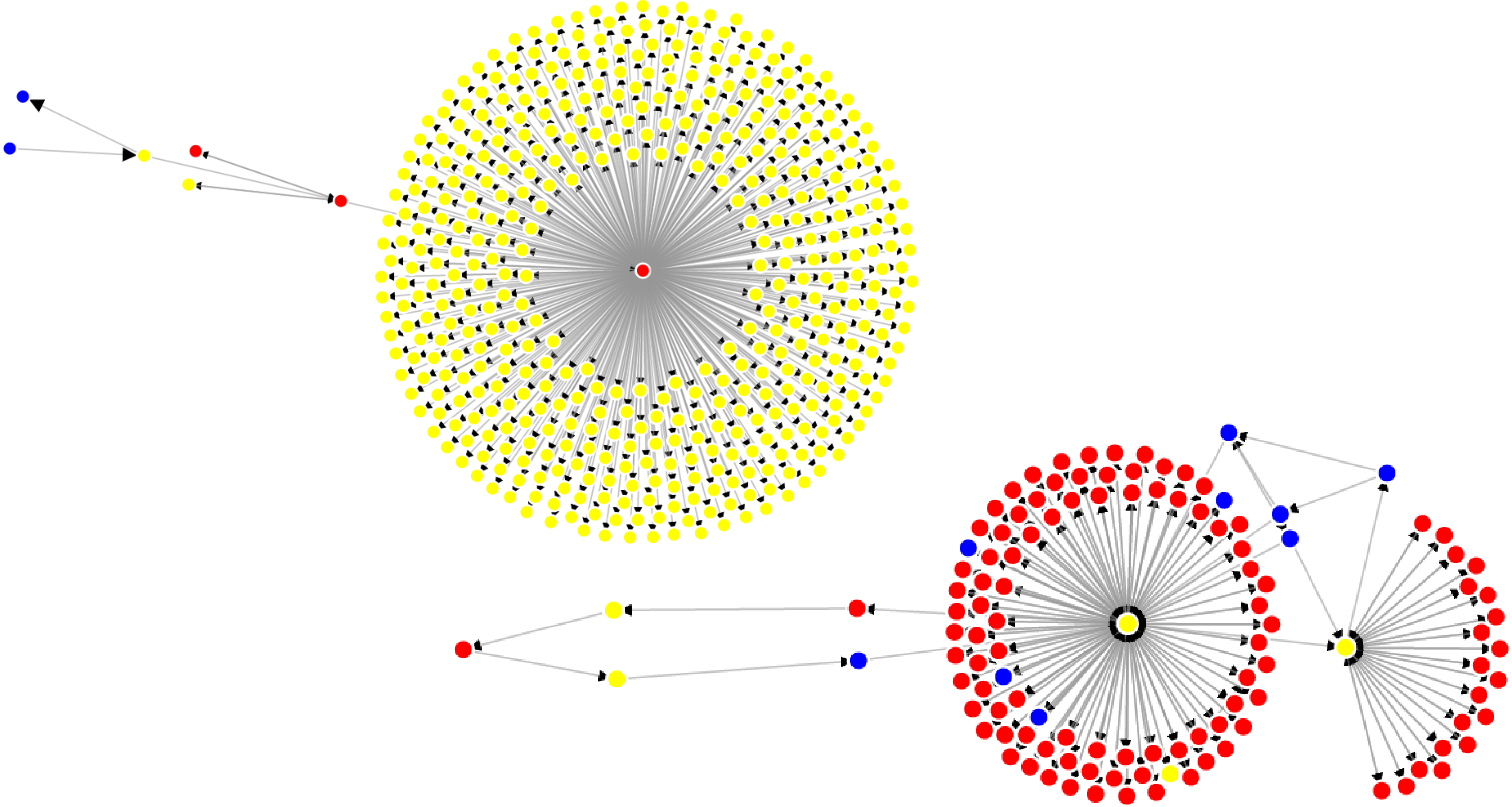
Address: 1HKywxiL4JziqXrzLKhmB6a74ma6kxbSDj

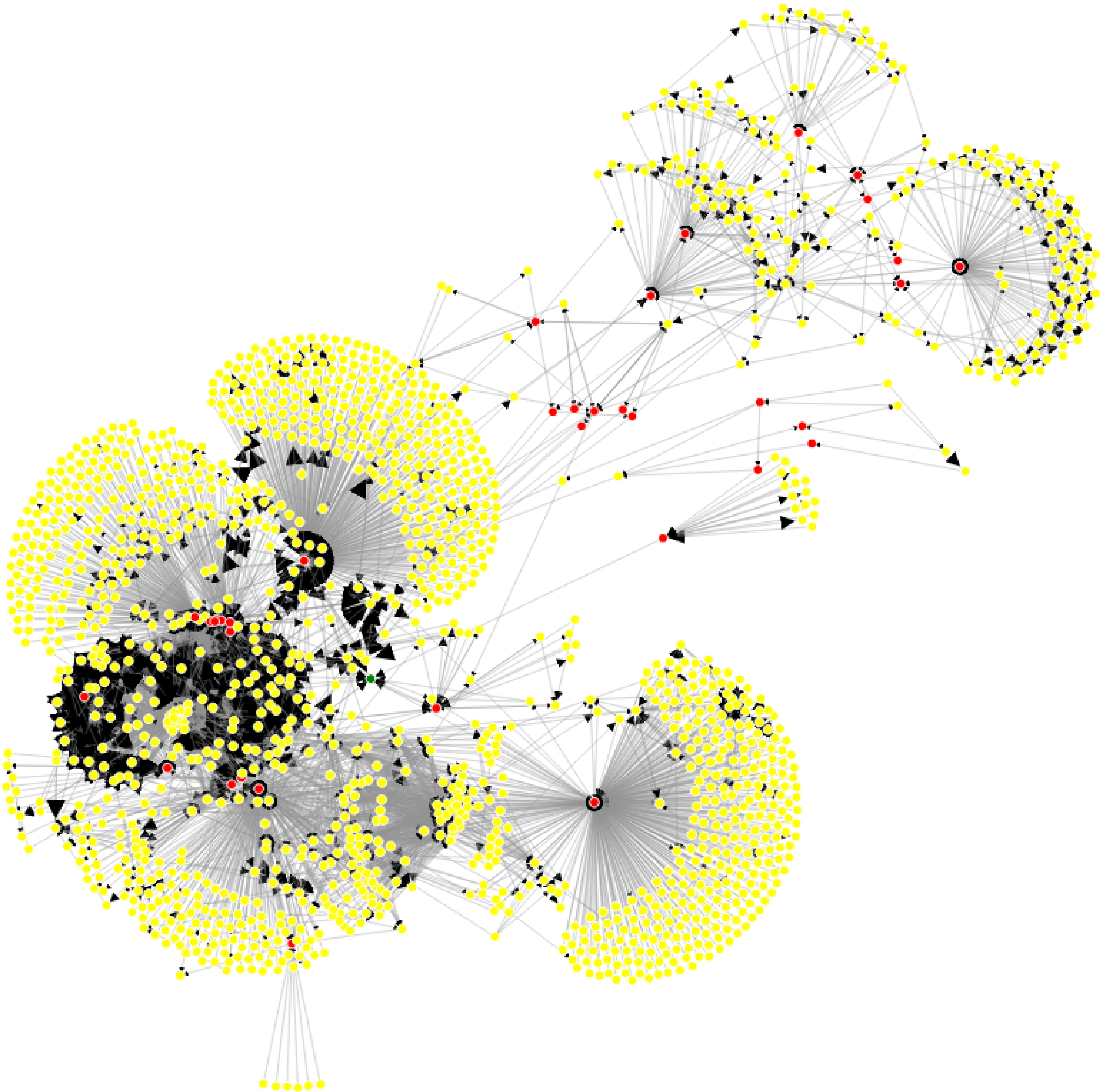
March to October 2013: 59 BTC ≈ 23000 USD has been stolen from 10 of these addresses

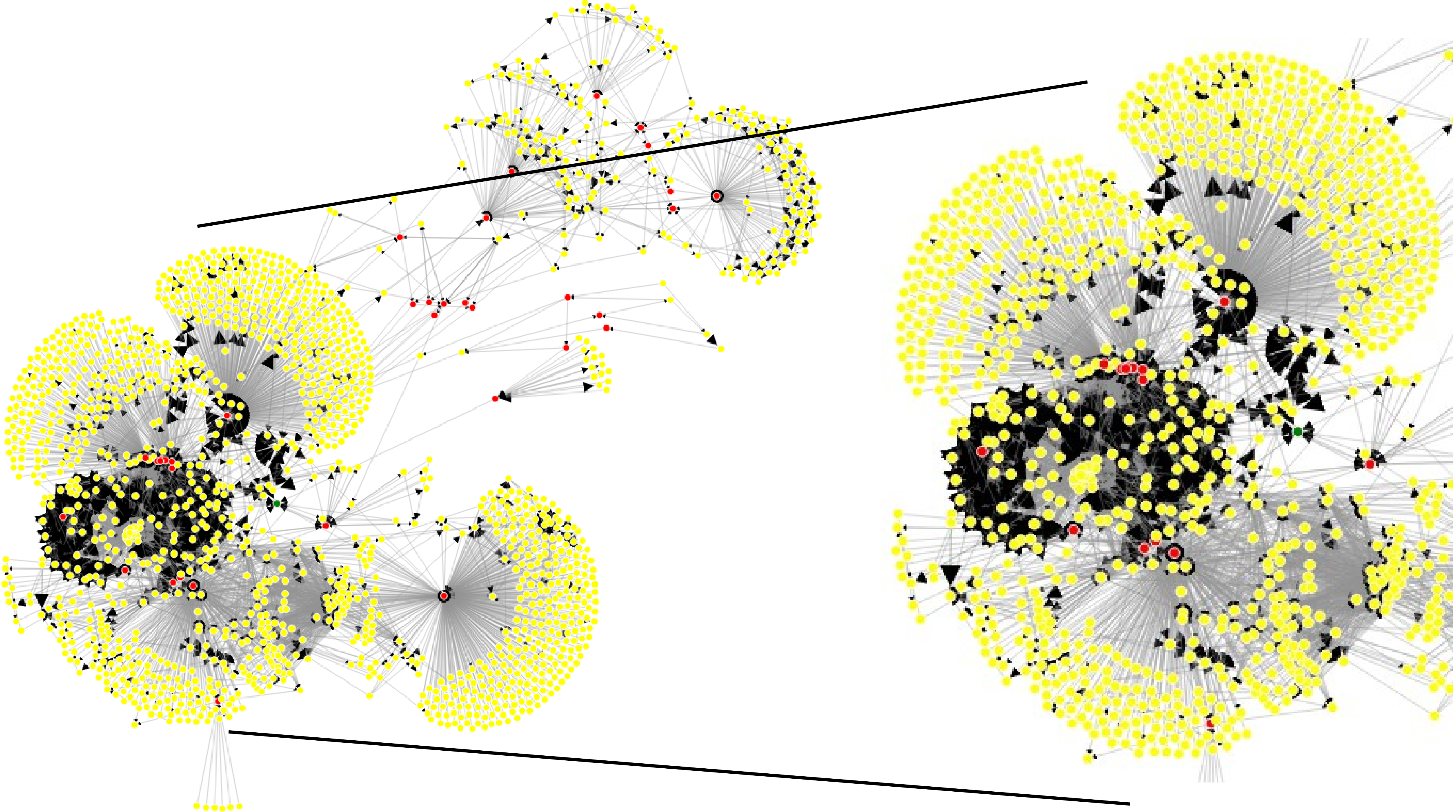
Possible cause

Poor entropy? At least 3 keys are known to be generated by implementations with Javascript's RNG problem









Unspendable Bitcoins

Recall:

- HASH160 = RIPEMD160(SHA256(*K*))
- Bitcoin address = $\mathrm{base58}\left(0\mathrm{x}00 \parallel \mathrm{HASH160} \parallel \left\lfloor \frac{\mathrm{SHA256}(\mathrm{SHA256}(0\mathrm{x}00 \parallel \mathrm{HASH160}))}{2^{224}} \right\rfloor\right)$

Idea

Transfer bitcoins to an account for which (*most likely*) no corresponding cryptographic key-pair exists

This results in deflation → increasing the value of other bitcoins

Unspendable Bitcoins

Recall:

- HASH160 = RIPEMD160(SHA256(*K*))
- Bitcoin address = $\mathrm{base58}\left(0\mathrm{x}00 \parallel \mathrm{HASH160} \parallel \left\lfloor \frac{\mathrm{SHA256}(\mathrm{SHA256}(0\mathrm{x}00 \parallel \mathrm{HASH160}))}{2^{224}} \right\rfloor\right)$

Idea

Transfer bitcoins to an account for which (*most likely*) no corresponding cryptographic key-pair exists

This results in deflation → increasing the value of other bitcoins

Can we give a lower bound on these unspendable / burned bitcoins?

Unspendable Bitcoins

HASH160	Bitcoin address	BTC
0000000000000000000000000000000000000000	1111111111111111111114oLvT2	2.94896715
0000000000000000000000000000000000000001	11111111111111111111BZbvjr	0.01000000
0000000000000000000000000000000000000002	11111111111111111111HeBAGj	0.00000001
0000000000000000000000000000000000000003	11111111111111111111QekFQw	0.00000001
0000000000000000000000000000000000000004	11111111111111111111UpYBrS	0.00000001
0000000000000000000000000000000000000005	11111111111111111111g4hiWR	0.00000001
0000000000000000000000000000000000000006	11111111111111111111jGyPM8	0.00000001
0000000000000000000000000000000000000007	11111111111111111111o9FmEC	0.00000001
0000000000000000000000000000000000000008	11111111111111111111ufYVpS	0.00000001
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	1GZQKjsC97yasxRj1wtYf5rC61AxpR1zmr	0.00012000
ffffffffffffffffffffffffffffffffffffffff	1QLbz7JHiBTspS962RLKV8GndWFwi5j6Qr	0.01000005
151 miscellaneous ASCII HASH160 values		1.32340175

Unspendable Bitcoins

HASH160	Bitcoin address	BTC
0000000000000000000000000000000000000000	1111111111111111111114oLvT2	2.94896715
0000000000000000000000000000000000000001	11111111111111111111BZbvjr	0.01000000
0000000000000000000000000000000000000002	11111111111111111111HeBAGj	0.00000001
0000000000000000000000000000000000000003	11111111111111111111QekFQw	0.00000001
0000000000000000000000000000000000000004	11111111111111111111UpYBrS	0.00000001
0000000000000000000000000000000000000005	11111111111111111111g4hiWR	0.00000001
0000000000000000000000000000000000000006	11111111111111111111jGyPM8	0.00000001
0000000000000000000000000000000000000007	11111111111111111111o9FmEC	0.00000001
0000000000000000000000000000000000000008	11111111111111111111ufYVpS	0.00000001
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	1GZQKjsC97yasxRj1wtYf5rC61AxpR1zmr	0.00012000
ffffffffffffffffffffffffffffffffffffffff	1QLbz7JHiBTspS962RLKV8GndWFwi5j6Qr	0.01000005
151 miscellaneous ASCII HASH160 values		1.32340175

Like graffiti in the Bitcoin block chain

69206861766520696e76697369626c6520676621 i have invisible gf!

48617070792062646179 2c20416e647269617500 Happy bday, Andriau

2174692064656c69706d6f6320796c6c616e6946 !ti delipmoc yllaniF

```
2d2d2d424547494e2054524942555445 2d2d2d20 ---BEGIN TRIBUTE---
232e2f4269744c656e20202020202020202020202020 #./BitLen
3a3a3a3a3a3a3a3a3a3a3a3a3a3a3a3a3a3a3a20 :::::::::::::::::::
3a3a3a3a3a3a3a2e3a3a2e3a3a2e3a2e3a3a3a20 :::::::.::.::.:.:::
3a2e3a203a2e272027202720272027203a203a20 :.: :.' ' ' ' ' : :
3a2e3a2727202c2c7869572c2234782c20272720 :.:'' ,,xiW,"4x, ''
3a20202c6457575758585858692c3457582c2020 :  ,dWWWXXXXi,4WX,
2720645757575858583722202020202060582c20 ' dWWWXXX7"     `X,
206c5757575858372020205f5f2020205f205820 lWWWXX7   __   _ X
3a575757585837202c785858372720225e5e5820 :WWWXX7 ,xXX7' "^^X
6c57575758372c205f2e2b2c2c205f2e2b2e2c20 lWWWX7, _.+,, _.+.,
3a575757372c2e20605e222d22202c5e2d272020 :WWW7,. `^"-" ,^-'
205757222c583a2020202020202020582c202020 WW",X:        X,
2022375e5e586c2e202020205f285f7837272020 "7^^Xl.    _(_x7'
206c2028203a583a202020202020205f5f205f20 l ( :X:       __ _
20602e2022205858202 02c787857575757583720 `. " XX  ,xxWWWWX7
202029582d2022222034582220 2e5f5f5f2e2020 )X- "" 4X" .___.
2c57205820202020203a586920205f2c2c5f2020 ,W X     :Xi  _,,_
5757205820202020202034586979585757586420 WW X      4XiyXWWXd
2222202c2c202020202020345857575757585820 "" ,,      4XWWWWXX
2c205237582c20202020202020225e3434375e20 , R7X,       "^447^
522c20223452586b2c2020202020205f2c202c20 R, "4RXk,      _, ,
54576b20202234525858692c20202058272c7820 TWk  "4RXXi,   X',x
6c54576b2c202022345252523727203420584820 lTWk,  "4RRR7' 4 XH
3a6c5757576b2c20205e22202020202060342020 :lWWWk,  ^"     `4
3a3a5454585757692c5f2020586c6c203a2e2e20 ::TTXWWi,_  Xll :..
3d2d3d2d3d2d3d2d3d2d3d2d3d2d3d2d3d2d3d20 =-=-=-=-=-=-=-=-=-=
4c454e20227261626269222053415353414d4120 LEN "rabbi" SASSAMA
2020202020313938302d32303131202020202020 1980-2011
4c656e20776173206f757220667269656e642e20 Len was our friend.
41206272696c6c69616e74206d696e642c202020 A brilliant mind,
61206b696e6420736f756c2c20616e6420202020 a kind soul, and
612064657669 6f757320736368656d65723b2020 a devious schemer;
68757362616e6420746f204d6572656469746820 husband to Meredith
62726f7468657220746f2043616c76696e2c2020 brother to Calvin,
736f6e20746f204a696d20616e64202020202020 son to Jim and
44616e61204861727473686f726e2c2020202020 Dana Hartshorn,
636f617574686f7220616e642020202020202020 coauthor and
636f666f756e64657220616e6420202020202020 cofounder and
53686d6f6f20616e6420736f206d756368202020 Shmoo and so much
6d6f72652e2020576520646564696361746520 20 more.  We dedicate
7468697320736696c6c79206861636b20746f2020 this silly hack to
4c656e2c2077686f20776f756c64206861766520 Len, who would have
666f756e642069742061627 36f6c7574656c7920 found it absolutely
68696c6172696f75732e20202020202020202020 hilarious.
2d2d44616e204b616d696e736b792c2020202020 --Dan Kaminsky,
54726176697320476f6f64737065656420202020 Travis Goodspeed
```

69206861766520696e76697369626c6520676621 i have invisible gf!

4861707079206264 61792c20416e647269617500 Happy bday, Andriau

2174692064656c69706d6f6320796c6c616e6946 !ti delipmoc yllaniF

```
502e532e20204d792061706f6c6f676965732c20 P.S.  My apologies,
426974436f696e2070656f706c652e2020486520 BitCoin people.  He
616c736f20776f756c6420686176652020202020 also would have
4c4f4c27642061742042697443 6f696e27732020 LOL'd at BitCoin's
6e657720646570656e64656e63792075706f6e20 new dependency upon
202020415343494920424552 4e414e4b45202020 ASCII BERNANKE
3a273a3a2e3a3a3a3a3a2e3a3a3a2e3a3a2e3a20 :'::.:::::.:::.::.:
3a203a2e3a202720272027202720 3a203a273a20 : :.: ' ' ' ' : :':
3a2e3a20202020205f2e5f5f20202020272e3a20 :.:     _.__    '.:
3a2020205f2c5e22202020225e782c2020203a20 :   _,^"   "^x,   :
272020783727202020202020202060342c202020 '  x7'        `4,
2058583720202020202020202020202034585820 XX7            4XX
2058582020202020202020202020202020585820 XX              XX
20586c202c7878782c2020202c7878782c585820 Xl ,xxx,   ,xxx,XX
282027205f2c2b6f2c207c202c6f2b2c22202020 ( ' _,+o, | ,o+,"
2034202020222d5e27205820225e2d2722203720 4   "-^' X "^-'" 7
206c2c2020202020282029292020202020 2c5820 l,     ( ))     ,X
203a58782c5f202c785858587878 2c5f2c585820 :Xx,_ ,xXXXxx,_,XX
20203458586958272d5f5f5f2d60585858582720 4XXiX'-___-`XXXX'
202034585869 2c5f2020205f6958583727202020 4XXi,_   _iXX7'
20202c2060345858585858585858585e205f2c20 , `4XXXXXXXXX^ _,
202058782c20202222 5e5e5e5858372c78582020 Xx,  ""^^^XX7,xX
572c22345757782c5f205f2c58785757583727 20 W,"4WWx,_ _,XxWWX7'
5877692c20223457573722223457573727 2c5720 Xwi, "4WW7""4WW7',W
54585857772c205e3720586b203437202c574820 TXXWw, ^7 Xk 47 ,WH
3a5458585857772c5f2022292c202c7757543a20 :TXXXWw,_ "), ,wWT:
3a3a54545858575757206c586c205757543a2020 ::TTXXWWW lXl WWT:
2d2d2d2d454e4420545249425554452d2d2d2d20 ----END TRIBUTE----
```

Unspendable Bitcoins

- If P is the **point at infinity**, then it is represented by the single byte 00.
- An **uncompressed point** starts with the byte 04 followed by the 256-bit x- and 256-bit y-coordinate of the point: $04 \parallel x \parallel y$ $(= 2\lceil \log_2(p) \rceil + 1$ bytes).
- A point is **compressed** by first computing a parity bit b of the y-coordinate as $b = (y \bmod 2) + 2$ and converting this to a byte value $\mathrm{b} \parallel x$ $(= \lceil \log_2(p) \rceil + 1$ bytes).

Unspendable Bitcoins

- If P is the **point at infinity**, then it is represented by the single byte 00.
- An **uncompressed point** starts with the byte 04 followed by the 256-bit x- and 256-bit y-coordinate of the point: $04 \parallel x \parallel y$ $(= 2\lceil \log_2(p) \rceil + 1$ bytes).
- A point is **compressed** by first computing a parity bit b of the y-coordinate as $b = (y \bmod 2) + 2$ and converting this to a byte value $\mathrm{b} \parallel x$ $(= \lceil \log_2(p) \rceil + 1$ bytes).

Public key	Valid encoding	Point on curve	Bitcoin address	Balance in BTC
00	✓	✓	1FYMZEHnszCHKTBdFZ2DLrUuk3dGwYKQxh	2.08000002
$0\overset{128}{\dots}0$	✗	✗	13VmALKHkCdSN1JULkP6RqW3LcbpWvgryV	0.00010000
$040\overset{126}{\dots}0$	✓	✗	16QaFeudRUt8NYy2yzjm3BMvG4xBbAsBFM	0.01000000

Unspendable Bitcoins

- If P is the **point at infinity**, then it is represented by the single byte 00.
- An **uncompressed point** starts with the byte 04 followed by the 256-bit x- and 256-bit y-coordinate of the point: $04 \parallel x \parallel y$ $(= 2\lceil\log_2(p)\rceil + 1$ bytes).
- A point is **compressed** by first computing a parity bit b of the y-coordinate as $b = (y \bmod 2) + 2$ and converting this to a byte value $\mathrm{b} \parallel x$ $(= \lceil\log_2(p)\rceil + 1$ bytes).

Public key	Valid encoding	Point on curve	Bitcoin address	Balance in BTC
$\emptyset$	✗	✗	1HT7xU2Ngenf7D4yocz2SAcnNLW7rK8d4E	68.80080003
00	✓	✓	1FYMZEHnszCHKTBdFZ2DLrUuk3dGwYKQxh	2.08000002
$0\overset{128}{\dots}0$	✗	✗	13VmALKHkCdSN1JULkP6RqW3LcbpWvgryV	0.00010000
$04\,0\overset{126}{\dots}0$	✓	✗	16QaFeudRUt8NYy2yzjm3BMvG4xBbAsBFM	0.01000000

Conclusions

✓ **ECC is well-deployed and used in practice**

Statistics

Elliptic curves are used in practice

- > 1 out of 10 in SSH
- > 1 out of 14 in TLS
- 100% of all keys in Bitcoin
- However, hosts prefer lower computation and bandwidth costs over increased security

Conclusions

✓ **ECC is well-deployed and used in practice**

Statistics

Elliptic curves are used in practice

- > 1 out of 10 in SSH
- > 1 out of 14 in TLS
- 100% of all keys in Bitcoin
- However, hosts prefer lower computation and bandwidth costs over increased security

✓ **ECC is not immune to insufficient entropy and software bugs**

Cryptographic sanity check

- We found many instances of repeated public SSH and TLS keys
- Bitcoin: there are many signatures sharing ephemeral nonces
  This lead to the theft of a at least 59 BTC
- Bitcoin: > 75 BTC $\approx$ 14000 USD is unspendable