Elliptic Curve Cryptography in Practice

Joppe W. Bos

Joint work with
J. Alex Halderman, Nadia Heninger, Jonathan Moore,
Michael Naehrig, Eric Wustrow
300 BC: Euclid studies conics
262-190 BC: Apollonius of Perga, *On conics*, introducing the name "ellipse"
200-300: Diophantus of Alexandria, *Arithmetica*

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

300 BC: Euclid studies conics

262-190 BC: Apollonius of Perga, *On conics*, introducing the name "ellipse"

200-300: Diophantus of Alexandria, *Arithmetica*

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

**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

Elliptic Curves – An Incomplete Historic Overview

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*

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

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

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

1901: Weierstrass, addition formula for elliptic functions to the addition of points on cubic curves

1901: Poincaré, tied all these ideas together: elliptic curves field as we know it

1933: Hasse, estimate of the number of points on an elliptic curve
\[ |\#E(\mathbb{F}_p) - (p + 1)| \leq 2\sqrt{p} \]

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*
1933: Hasse, estimate of the number of points on an elliptic curve
\[ |\#E(F_p) - (p + 1)| \leq 2\sqrt{p} \]

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
1933: Hasse, estimate of the number of points on an elliptic curve
\[ |\#E(F_p) - (p + 1)| \leq 2\sqrt{p} \]

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

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

<table>
<thead>
<tr>
<th>( p )</th>
<th>NIST FIPS 186-4</th>
<th>Certicom SEC1</th>
<th>OpenSSL</th>
<th>( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{192} - 2^{64} - 1 )</td>
<td>P-192</td>
<td>secp192r1</td>
<td>prime192v1</td>
<td>-3</td>
</tr>
<tr>
<td>( 2^{224} - 2^{96} + 1 )</td>
<td>P-224</td>
<td>secp224r1</td>
<td>secp224r1</td>
<td>-3</td>
</tr>
<tr>
<td>( 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 )</td>
<td>P-256</td>
<td>secp256r1</td>
<td>prime256v1</td>
<td>-3</td>
</tr>
<tr>
<td>( 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 )</td>
<td>P-384</td>
<td>secp384r1</td>
<td>secp384r1</td>
<td>-3</td>
</tr>
<tr>
<td>( 2^{521} - 1 )</td>
<td>P-521</td>
<td>secp521r1</td>
<td>secp521r1</td>
<td>-3</td>
</tr>
<tr>
<td>( 2^{256} - 2^{32} - 977 )</td>
<td></td>
<td>secp256k1</td>
<td>secp256k1</td>
<td>0</td>
</tr>
</tbody>
</table>
Elliptic Curve Public Key Pairs

$(d, Q)$ such that $d \in \mathbb{F}_p^\times, E(\mathbb{F}_p) \ni Q = dG$
Elliptic Curve Public Key Pairs

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

Elliptic Curve Key Exchange

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

\[ P = d_aQ_b = d_bQ_a \]
Elliptic Curve Public Key Pairs

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

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 \((d, Q, m)\)

\(k \in \mathbb{F}_n^\times, \quad kG = (x, y), \quad r = x \mod n\)

\(s = k^{-1}(\text{Hash}(m) + dr) \mod n, \quad \text{Signature: } (r, s)\)
Elliptic Curve Public Key Pairs

\[(d, Q)\] such that \(d \in \mathbb{F}_n^\times, E(\mathbb{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 \((d, Q, m)\)

\[k \in \mathbb{F}_n^\times, \quad kG = (x, y), \quad r = x \mod n\]

\[s = k^{-1}(\text{Hash}(m) + dr) \mod n, \quad \text{Signature: } (r, s)\]

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 Curve Public Key Pairs

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

Elliptic Curve Key Exchange

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

\[ P = d_aQ_b = d_bQ_a \]

Elliptic Curve Digital Signatures \((d, Q, m)\)

\[ k \in \mathbb{F}_n^\times, \quad kG = (x, y), \quad r = \frac{x}{\text{mod } n} \]
\[ s = k^{-1}(\text{Hash}(m) + dr) \mod n, \quad \text{Signature: } (r, s) \]

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

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

\[ d \equiv r^{-1}(ks - \text{Hash}(m)) \mod n \]
Elliptic Curves in Cryptography - II

Elliptic Curve Public Key Pairs

\((d, Q)\) such that \(d \in \mathbb{F}_n^x, E(\mathbb{F}_p) \ni Q = dG\)

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

\((d, Q, m)\)

\(k \in \mathbb{F}_n^x, \quad kG = (x, y), \quad r = x \mod n\)

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

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

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

\(d \equiv r^{-1}(ks - \text{Hash}(m)) \mod n\)

Secp256k1: A special curve

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

\(\psi: E \rightarrow E, (x, y) \rightarrow (\zeta x, -y)\)

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

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]

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)
“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)
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

“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)

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)

### Secure Shell (SSH) - Statistics

Total cipher suites: 12,114,534

- ECDSA
  - 1,249,273 (10.3%) supported
    - ecdsa-sha2-nistp256
    - ecdsa-sha2-nistp384
    - ecdsa-sha2-nistp521

"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)

---

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

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.”
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.”

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.”

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
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
Idea:

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

repeat {
  connect to server with $L$
  if answer $a \neq \emptyset$ write down curve info
  $L = L \setminus \{a\}$
} until $a == \emptyset$
Idea:

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

repeat {
    connect to server with $L$
    if answer $a \neq \emptyset$ write down curve info
    $L = L \setminus \{a\}$
} until $a == \emptyset$

Total hosts: 30.2M

ECDH(E) 2.2M (7.2%)

98% supported nistp256

80% supported nistp384

17% supported nistp521
Transport Layer Security (TLS) - Statistics

Idea:
$L = \{\text{a set of 38 ECDH and ECDHE cipher suites (28 different curves)}\}$

\[
\text{repeat } \{
\begin{align*}
&\text{connect to server with } L \\
&\text{if answer } a \neq \emptyset \text{ write down curve info} \\
&L = L \setminus \{a\}
\end{align*}
\} \text{ until } a = \emptyset
\]

- 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:
$L = \text{a set of 38 ECDH and ECDHE cipher suites (28 different curves)}$
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 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”

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”


- 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 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"  


Bitcoin address is not really an ECDSA key $K$

$\text{HASH160} = \text{RIPEMD160}(\text{SHA256}(K))$

$\text{Bitcoin address} = \text{base58}(0x00 \parallel \text{HASH160} \parallel \frac{\text{SHA256}(\text{SHA256}(0x00\parallel\text{HASH160}))}{2^{224}})$
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”


Bitcoin address is not really an ECDSA key $K$

$$\text{HASH160} = \text{RIPEMD160} (\text{SHA256}(K))$$

$$\text{Bitcoin address} = \text{base58} (0x00 \parallel \text{HASH160} \parallel \frac{\text{SHA256} (\text{SHA256} (0x00 \parallel \text{HASH160}))}{2^{224}})$$

**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 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" 


Bitcoin address is not really an ECDSA key $K$

$$\text{HASH160} = \text{RIPEMD160(SHA256(K))}$$

$$\text{Bitcoin address} = \text{base58}(0x00 \parallel \text{HASH160} \parallel \text{SHA256(0x00||SHA256(\text{SHA256(0x00||HASH160)}))}_{2^{224}})$$

**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


- 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
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

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


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
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.


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

2013: Question #2

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

\[ Q = dG \] poor randomness might result in repeated \( d \)

<table>
<thead>
<tr>
<th>RSA</th>
<th>( p_1q_1 )</th>
<th>( p_1q_2 )</th>
<th>( p_2q_1 )</th>
<th>( p_2q_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1q_1 )</td>
<td>?</td>
<td>Insecure</td>
<td>Insecure</td>
<td>Secure</td>
</tr>
<tr>
<td>( p_1q_2 )</td>
<td>Insecure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_2q_1 )</td>
<td>Insecure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_2q_2 )</td>
<td>Secure</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ECC</th>
<th>( d_1 )</th>
<th>( d_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>?</td>
<td>Secure</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>Secure</td>
<td></td>
</tr>
</tbody>
</table>
Key Generation

\( Q = dG \) poor randomness might result in repeated \( d \)

<table>
<thead>
<tr>
<th>RSA</th>
<th>( p_1q_1 )</th>
<th>ECC</th>
<th>( d_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1q_1 )</td>
<td>?</td>
<td>( d_1 )</td>
<td>?</td>
</tr>
<tr>
<td>( p_1q_2 )</td>
<td>Insecure</td>
<td>( d_1 )</td>
<td>?</td>
</tr>
<tr>
<td>( p_2q_1 )</td>
<td>Insecure</td>
<td>( d_2 )</td>
<td>Secure</td>
</tr>
<tr>
<td>( p_2q_2 )</td>
<td>Secure</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \( d \)

<table>
<thead>
<tr>
<th>RSA</th>
<th>( p_1q_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1q_1 )</td>
<td>?</td>
</tr>
<tr>
<td>( p_1q_2 )</td>
<td>Insecure</td>
</tr>
<tr>
<td>( p_2q_1 )</td>
<td>Insecure</td>
</tr>
<tr>
<td>( p_2q_2 )</td>
<td>Secure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ECC</th>
<th>( d_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>?</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>Secure</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>SSH</th>
<th>TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td># elliptic curve public keys</td>
<td>1.2 million</td>
<td>5.4 million</td>
</tr>
<tr>
<td># unique keys</td>
<td>0.8 million</td>
<td>5.2 million</td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th></th>
<th>SSH</th>
<th>TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td># elliptic curve public keys</td>
<td>1.2 million</td>
<td>5.4 million</td>
</tr>
<tr>
<td># unique keys</td>
<td>0.8 million</td>
<td>5.2 million</td>
</tr>
</tbody>
</table>

### SSH

**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
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.
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
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:
- $\text{HASH160} = \text{RIPEMD160} (\text{SHA256}(K))$
- Bitcoin address = base58($0x00 \parallel \text{HASH160} \parallel \left\lceil \frac{\text{SHA256} (\text{SHA256}(0x00 \parallel \text{HASH160}))}{2^{224}} \right\rceil$)

**Idea**
Transfer bitcoins to an account for which (*most likely*) no corresponding cryptographic key-pair exists.
This results in deflation $\rightarrow$ increasing the value of other bitcoins.
Unspendable Bitcoins

Recall:
- $\text{HASH160} = \text{RIPEMD160}(\text{SHA256}(K))$
- Bitcoin address = base58($0\times00 \parallel \text{HASH160} \parallel \frac{\text{SHA256}(\text{SHA256}(0\times00\parallel \text{HASH160}))}{2^{224}}$)

Idea
Transfer bitcoins to an account for which (most likely) no corresponding cryptographic key-pair exists. This results in deflation $\rightarrow$ increasing the value of other bitcoins.

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

<table>
<thead>
<tr>
<th>HASH160</th>
<th>Bitcoin address</th>
<th>BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000000000000000000000000000</td>
<td>1111111111111111111114oLvT2</td>
<td>2.94896715</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000001</td>
<td>11111111111111111111BZbvjr</td>
<td>0.01000000</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000002</td>
<td>11111111111111111111HeBAGj</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000003</td>
<td>11111111111111111111QekFQw</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000004</td>
<td>11111111111111111111UpYBrS</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000005</td>
<td>11111111111111111111g4hiWR</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000006</td>
<td>11111111111111111111jGyPM8</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000007</td>
<td>11111111111111111111o9FmEC</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000000000008</td>
<td>11111111111111111111ufYVpS</td>
<td>0.00000001</td>
</tr>
<tr>
<td>aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa</td>
<td>1GZQKjsC97yasxRj1wtYf5rC61AxpR1zmr</td>
<td>0.00012000</td>
</tr>
<tr>
<td>ffffffffffffffffffffffffffffffffffffffff</td>
<td>1QLbz7JHiBTspS962RLKV8GndWFwi5j6Qr</td>
<td>0.01000005</td>
</tr>
<tr>
<td>151 miscellaneous ASCII HASH160 values</td>
<td></td>
<td>1.32340175</td>
</tr>
</tbody>
</table>
# Unspendable Bitcoins

<table>
<thead>
<tr>
<th>HASH160</th>
<th>Bitcoin address</th>
<th>BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000000000000000000000000000</td>
<td>1111111111111111111114oTvT2</td>
<td>2.94896715</td>
</tr>
<tr>
<td>0000000000000000000000000000000001</td>
<td>111111111111111111111BZbvjr</td>
<td>0.01000000</td>
</tr>
<tr>
<td>0000000000000000000000000000000002</td>
<td>111111111111111111111HeBAGj</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000003</td>
<td>111111111111111111111QekFQw</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000004</td>
<td>111111111111111111111UpYBrS</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000005</td>
<td>111111111111111111111g4hiWR</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000006</td>
<td>111111111111111111111jGyPM8</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000007</td>
<td>111111111111111111111o9FmEC</td>
<td>0.00000001</td>
</tr>
<tr>
<td>0000000000000000000000000000000008</td>
<td>111111111111111111111ufYVpS</td>
<td>0.00000001</td>
</tr>
<tr>
<td>aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa</td>
<td>1GZQKjRsC97yasxRj1wtYf5rC61AxpR1zmr</td>
<td>0.00012000</td>
</tr>
<tr>
<td>ffffffffffffffffffffffffffffffffffffffff</td>
<td>1QLbz7JHiBTspS962RLKV8GndWFwi5j6Qr</td>
<td>0.01000005</td>
</tr>
<tr>
<td>151 miscellaneous ASCII HASH160 values</td>
<td></td>
<td>1.32340175</td>
</tr>
</tbody>
</table>

Like graffiti in the Bitcoin block chain
I have invisible gf!

Happy bday, Andriau

!ti delipmoc yllaniF
people. He

cofounder and 

copyright and

Dana

son to Jim and

a devious schemer;

a kind soul, and

Len was our friend.

2011

Shmoo

BitCoin

LOL'd

BitCoin's

Andriau

46297443366966e2070e656706e552420201986520bitcoin people. He

also would have

4c4fc4f27c64206174204269744369e207732020 LOL'd at Bitcoin's

6e65772064657066e46566e637920750766e20 new dependency upon

20201941343449202425524422e4b5202020

ASCII BERNANKE

as "^x^",

xx, Xx, Xx, Xx

Xx, Xx, Xx, Xx

an invisible gf!

4861707092062461792c20416e647269617500 Happy bday, Andriau

2174692064656c69706d6f6320796c6c16e6946 !ti delipmoc yllaniF

50e5532e202d492d902617606f6c67695732c0 P.S. My apologies, 42697443366966e2070e656706e552420201986520bitcoin people. He

also would have

4c4fc4f27c64206174204269744369e207732020 LOL'd at Bitcoin's

6e65772064657066e46566e637920750766e20 new dependency upon

20201941343449202425524422e4b5202020

ASCII BERNANKE

as "^x^",

xx, Xx, Xx, Xx

Xx, Xx, Xx, Xx

an invisible gf!

4861707092062461792c20416e647269617500 Happy bday, Andriau

2174692064656c69706d6f6320796c6c16e6946 !ti delipmoc yllaniF

50e5532e202d492d902617606f6c67695732c0 P.S. My apologies, 42697443366966e2070e656706e552420201986520bitcoin people. He

also would have

4c4fc4f27c64206174204269744369e207732020 LOL'd at Bitcoin's

6e65772064657066e46566e637920750766e20 new dependency upon

20201941343449202425524422e4b5202020

ASCII BERNANKE

as "^x^",

xx, Xx, Xx, Xx

Xx, Xx, Xx, Xx

an invisible gf!
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: \( 04 \parallel x \parallel y \) (= \( 2[\log_2(p)] + 1 \) bytes).

A point is **compressed** by first computing a parity bit \( b \) of the \( y \)-coordinate as \( b = (y \mod 2) + 2 \) and converting this to a byte value \( b \parallel x \) (= \( \lceil \log_2(p) \rceil + 1 \) 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 $b = (y \mod 2) + 2$ and converting this to a byte value $b \parallel x$ ($= \lceil \log_2(p) \rceil + 1$ bytes).

<table>
<thead>
<tr>
<th>Public key</th>
<th>Valid encoding</th>
<th>Point on curve</th>
<th>Bitcoin address</th>
<th>Balance in BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>✔</td>
<td>✔</td>
<td>1FYMZEHnszCHKTBdFZ2DLrUuk3dGwYKQxh</td>
<td>2.08000002</td>
</tr>
<tr>
<td>01280</td>
<td>✗</td>
<td>✗</td>
<td>13VmALKHkCdSN1JULkP6RqW3LcbpWvgryV</td>
<td>0.00010000</td>
</tr>
<tr>
<td>0401260</td>
<td>✔</td>
<td>✗</td>
<td>16QaFeudRUt8NYy2yzjm3BMvG4xBbAsBFM</td>
<td>0.01000000</td>
</tr>
</tbody>
</table>
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 \( \parallel \) \( x \) \( \parallel \) \( y \) (\( = 2\left\lceil \log_2(p) \right\rceil + 1 \) bytes).
- A point is **compressed** by first computing a parity bit \( b \) of the \( y \)-coordinate as \( b = (y \mod 2) + 2 \) and converting this to a byte value \( b \parallel x \) (\( = \left\lceil \log_2(p) \right\rceil + 1 \) bytes).

<table>
<thead>
<tr>
<th>Public key</th>
<th>Valid encoding</th>
<th>Point on curve</th>
<th>Bitcoin address</th>
<th>Balance in BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>✗</td>
<td>✗</td>
<td>1HT7xU2Ngenf7D4yocz2SAcnNLW7rK8d4E</td>
<td>68.80080003</td>
</tr>
<tr>
<td>00</td>
<td>✓</td>
<td>✓</td>
<td>1FYMZEHnszCHKTBdFZ2DLrUuk3dGwYKQxh</td>
<td>2.08000002</td>
</tr>
<tr>
<td>0^{128}0</td>
<td>✗</td>
<td>✗</td>
<td>13VmALKHkCdSN1JULkP6RqW3LcbpWvgryV</td>
<td>0.00010000</td>
</tr>
<tr>
<td>040^{126}0</td>
<td>✓</td>
<td>✗</td>
<td>16QaFeudRUt8NYy2yzjm3BMvG4xBbAsBFM</td>
<td>0.01000000</td>
</tr>
</tbody>
</table>
Conclusions

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

<table>
<thead>
<tr>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curves are used in practice</td>
</tr>
<tr>
<td>- &gt; 1 out of 10 in SSH</td>
</tr>
<tr>
<td>- &gt; 1 out of 14 in TLS</td>
</tr>
<tr>
<td>- 100% of all keys in Bitcoin</td>
</tr>
<tr>
<td>- However, hosts prefer lower computation and bandwidth costs over increased security</td>
</tr>
</tbody>
</table>
Conclusions

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

<table>
<thead>
<tr>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curves are used in practice</td>
</tr>
<tr>
<td>• &gt; 1 out of 10 in SSH</td>
</tr>
<tr>
<td>• &gt; 1 out of 14 in TLS</td>
</tr>
<tr>
<td>• 100% of all keys in Bitcoin</td>
</tr>
<tr>
<td>• However, hosts prefer lower computation and bandwidth costs over increased security</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Cryptographic sanity check</th>
</tr>
</thead>
<tbody>
<tr>
<td>• We found many instances of repeated public SSH and TLS keys</td>
</tr>
<tr>
<td>• Bitcoin: there are many signatures sharing ephemeral nonces</td>
</tr>
<tr>
<td>This lead to the theft of a at least 59 BTC</td>
</tr>
<tr>
<td>• Bitcoin: &gt; 75 BTC $\approx$ 14000 USD is unspendable</td>
</tr>
</tbody>
</table>