Digital Signatures

Thanks to Ari Juels for parts of this deck!
How digital signatures work: An analogy

- Suppose you had an ideal signet ring, i.e.,
  - Everyone knows what your seal looks like
    - “Public key” $PK$
  - But it can only be produced by someone with your ring
    - “Private key” $SK$
- Anyone can verify authenticity of seal $Sig$ on message $M$, but only holder of rabbit ring can create one
  - Rabbit seal proves ring holder signed message
Digital signatures: Technical view

Private: 011101

KeyGen

Public: 110011

SK

PK
Digital signatures: Technical view

Private

SK

Sign

(M, Sig)

M = 0100101

M = 0100101
Digital signatures: Technical view

Verify

PK

110011

Public

(M, Sig)

M = 0100101

✓ M was signed with SK

✗ Bad signature
Digital signatures: Technical view

KeyGen

Sign

Verify

M = 011101

SK

PK

Public

Private

M = 0100101

M = 0100101

(M, Sig)

M was signed with SK

Bad signature
Digital signatures

- Use to achieve security goal of \textit{message integrity} (prevent tampering)
What we want from digital signatures

message $m$

Alice's private key: $SK$

$m, S = \text{Sign}(SK, m)$

Verify($PK, m, S$)

- Security property:
  - Should only be possible to sign using $SK$, even though:
    - Alice's public key $PK$ is published, i.e., known to world;
    - Anyone can verify using $PK$. 
Schnorr identification ("interactive signature") scheme

Goal: Alice identifies herself to Bob by proving knowledge of her private key.

Private / Public Keys:
(a, A = g^a)

Alice's public key: A
First try

Private / Public Keys:
\((a, A = g^a)\)

Alice's public key: \(A\)

\[ s = ca \]

\[ g^s = A^c \]

✔︎
Better approach

Private / Public Keys: 
\((a, A = g^a)\)

One-time private key: \(r\)

Blinding factor: \(R = g^r\)

\[ R, s = ca + r \]

\[ g^s = RA^c \]

Intuition:
- Bob can verify that \(a\) is properly “mixed in”—so it’s really Alice.
- Alice removes \(a\) from exponent space—reveals it in \(s\), but…
- \(r\) is a one-time value that blinds, i.e., conceals \(a\).
Why isn't this a signature scheme?

Private / Public Keys: 
\( (a, A = g^a) \)

One-time private key: \( r \)

One-time blinding factor: \( R = g^r \)

\( R \), \( s = ca + r \)

\( g^s \overset{?}{=} RA^c \)

- Requires interaction
- Bob can't prove to another person that Alice "signed" \( c \)
Where might you use an identification scheme?

- What’s the benefit of using public-key, rather than symmetric-key cryptography?
- Why not use secret $K$ shared by phone and building?
Schnorr signature scheme

Private / Public Keys: 
\((a, A = g^a)\)

One-time private key: \(r\)

Public key: \(R = g^r\)

\[ c = H(R, m) \]

Signature:
\((R, s = ca + r)\)

Verification:
\[ g^s \overset{?}{=} RA^c \]

Intuition:
- Generate “challenge” \(c\) from \(m\)
Schnorr signature scheme

message \( m \)

Private / Public Keys:
\((a, A = g^a)\)

One-time
private key: \( r \)

Public key: \( R = g^r \)

\[ c = H(R, m) \]

Signature:
\((R, s = ca + r)\)

\[ g^s \overset{?}{=} RA^c \]

Verification

Very similar to
ECDSA ("Elliptic-Curve Digital Signature Algorithm"),
which is much more widely used
Schnorr signature scheme

message $m$

Private / Public Keys: $(a, A = g^a)$

One-time private key: $r$

Public key: $R = g^r$

$c = H(R, m)$

Signature: $(R, s = ca + r)$

Alice’s public key: $A$

$g^s \equiv RA^c$

Verification

Does this really have to be one-time?
The Sony PS3 break

- PS3 used ECDSA for code signing
  - (Who’s signing and who’s verifying?)
  - Box only accepts code signed by Sony.
  - (Why is this better than use of secret key / MAC?)
- In late 2010, the crypto in the PS3 was broken
  - George “Geohot” Hotz and Fail0verflow team, Dec. 2010
- How did it happen?
  - Sony used the same “one-time” \( r \) in every signature.
The Sony PS3 break

- Simplified version:
  - George got two sigs.:
    - \( s_1 = c_1 a + r \)
    - \( s_2 = c_2 a + r \)
    - \( s_1 - s_2 = (c_1 - c_2)a \)
  - \( a = \frac{s_1 - s_2}{c_1 - c_2} \)
  - Game over!

- Literally! (Perhaps minus comment)
Android Security Vulnerability
11 August 2013

What happened

We recently learned that a component of Android responsible for generating secure random numbers contains critical weaknesses, that render all Android wallets generated to date vulnerable to theft. Because the problem lies with Android itself, this problem will affect you if you have a wallet generated by any Android app. An incomplete list would be Bitcoin Wallet, blockchain.info wallet, BitcoinSpinner and Mycelium Wallet. Apps where you don’t control the private keys at all are not affected. For example, exchange frontends like the Coinbase or Mt Gox apps are not impacted by this issue because the private keys are not generated on your Android phone.
Where are digital signatures used?

- Code signing
- Bitcoin
  - Wallets / transaction signing (ECDSA)
- Authentication
  - E.g., EMV
Where are digital signatures used?
**Comparison:**

**RSA sigs. vs. ECDSA**

<table>
<thead>
<tr>
<th>Pros</th>
<th>RSA signatures</th>
<th>ECDSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factoring hardness well studied</td>
<td>• Factoring hardness well studied</td>
<td>• Fast signing</td>
</tr>
<tr>
<td>Widespread use</td>
<td>• Widespread use</td>
<td>• Short signatures (448-bit)</td>
</tr>
<tr>
<td>Fast verification</td>
<td>• Fast verification</td>
<td>• Short keys (224-bit*)</td>
</tr>
<tr>
<td>Limited use: iMessage, Bitcoin, etc.</td>
<td></td>
<td>• Limited use: iMessage, Bitcoin, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cons</th>
<th>RSA signatures</th>
<th>ECDSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow signing</td>
<td>• Slow signing</td>
<td>• EC DL problem not well studied</td>
</tr>
<tr>
<td>Long signatures (2048-bit)</td>
<td>• Long signatures (2048-bit)</td>
<td>• Slower verification than RSA</td>
</tr>
<tr>
<td>Long keys (2048-bit*)</td>
<td>• Long keys (2048-bit*)</td>
<td></td>
</tr>
</tbody>
</table>

*NIST recommended key length for years 2010-2030*
Certificates authorities and public-key infrastructure (PKI)

• Idea: Certificates are signed by Certificate Authorities (CA)
Certificates authorities and public-key infrastructure (PKI)

- Certificate Authority (CA) issues certificates to owners of domain names

\[ \text{Certificate } \text{ please on } pk? \]

\[ \text{cert}_{\text{BB}} = \text{Sig}(PK, m) \]

\[ m = (pk, \text{bigbank.com}) \]
Certificates authorities and public-key infrastructure (PKI)

- Certificate is verified by browser against CA key

Alice

- $\text{cert}_{\text{BB}}$ matches URL, $pk$
- $\text{Ver}(PK, \text{cert}_{\text{BB}}), \checkmark$

Prevents man-in-the-middle attacks
The site's security certificate is not trusted!

You attempted to reach blackboard.secure.force.com, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Google Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

You should not proceed, especially if you have never seen this warning before for this site.

Proceed anyway  Back to safety

Help me understand
Certificate hierarchy

- 50+ root certificates in most major browsers
- 650+ CAs globally
  - EFF Observatory
- X.509 is predominant standard