SplitKey – A Threshold Cryptography Case Study

Dr. Aivo Kalu – Security Engineer
Maximiliaan van de Poll – Product Head

March 13th 2019
Company introduction and background

- R&D intensive ICT company in Estonia
  - Research applied to practical security solutions since 1996
  - Researched time stamping, PKI, digital signatures, multi-party computation, ...
  - Developed and maintains Estonia’s X-Road (UXP), i-voting, Sharemind, ...
  - Research and development projects funded mostly by Estonian government and companies, EU H2020, USA DARPA and NATO
  - About 140 employees, 15% of them with PhDs
Company introduction and background

- R&D intensive ICT company in Estonia
  - Research applied to practical security solutions since 1996
  - Researched time stamping, PKI, digital signatures, multi-party computation, ...
  - Developed and maintains Estonia’s X-Road (UXP), i-voting, Sharemind, ...
  - Research and development projects funded mostly by Estonian government and companies, EU H2020, USA DARPA and NATO
  - About 140 employees, 15% of them with PhDs

- Estonia/Latvia/Lithuania so far had three widely used methods of authentication
  - ID-cards (smart-cards), Mobile-ID (SIM based), and one-time code cards
  - 2014, EU PSD2 regulation came with strong authentication demand
  - There was a market need for new kind of approach
SplitKey digital signature scheme

- Software-based 2-out-of-2 threshold cryptosystem
- Based on:
  - Camenisch, J., Lehmann, A., Neven, G., Samelin, K.: Virtual Smart Cards: How to sign with a password and a server. (2016)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \quad Gen(k)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d_1' \text{ Gen}(k), \ d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d_1' \text{ Gen}(k)\), \(d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\) \(\text{Gen}(k)\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\)  \(\text{Gen}(k)\)
- Client’s private key sharing: \(d_1' \text{ Gen}(k), \quad d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\)  \(\text{Gen}(k)\)
- Composite public key generation: \(n = n_1 \cdot n_2\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d'_1 \text{ Gen}(k), \quad d'_1 + d''_1 \equiv d_1 \pmod{\varphi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\) \(\text{Gen}(k)\)
- Composite public key generation: \(n = n_1 \cdot n_2\)

- Client’s part of the signature share: \(s'_1 = m^{d'_1} \pmod{n_1}\)
- Server’s part of the signature share: \(s''_1 = m^{d''_1} \pmod{n_1}\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d_1' \text{ Gen}(k), \ d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\) \(\text{Gen}(k)\)
- Composite public key generation: \(n = n_1 \cdot n_2\)
- Client’s part of the signature share: \(s_1' = m^{d_1'} \pmod{n_1}\)
- Server’s part of the signature share: \(s_1'' = m^{d_1''} \pmod{n_1}\)
- Client’s signature share: \(s_1 = s_1' \cdot s_1'' \pmod{n_1}\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d'_1 \text{ Gen}(k), \quad d'_1 + d''_1 \equiv d_1 \pmod{\phi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\) \(\text{Gen}(k)\)
- Composite public key generation: \(n = n_1 \cdot n_2\)

- Client’s part of the signature share: \(s'_1 = m^{d'_1} \pmod{n_1}\)
- Server’s part of the signature share: \(s''_1 = m^{d''_1} \pmod{n_1}\)
- Client’s signature share: \(s_1 = s'_1 \cdot s''_1 \pmod{n_1}\)
- Server’s signature share: \(s_2 = m^{d_2} \pmod{n_2}\)
SplitKey key pair generation and signing operation

- Client’s key pair generation: \((d_1, e), (n_1, e)\) \(\text{Gen}(k)\)
- Client’s private key sharing: \(d'_1\) \(\text{Gen}(k)\), \(d'_1 + d''_1 \equiv d_1 \pmod{\varphi(n_1)}\)
- Server’s key pair generation: \((d_2, e), (n_2, e)\) \(\text{Gen}(k)\)
- Composite public key generation: \(n = n_1 \cdot n_2\)

- Client’s part of the signature share: \(s'_1 = m^{d'_1} \pmod{n_1}\)
- Server’s part of the signature share: \(s''_1 = m^{d''_1} \pmod{n_1}\)
- Client’s signature share: \(s_1 = s'_1 \cdot s''_1 \pmod{n_1}\)
- Server’s signature share: \(s_2 = m^{d_2} \pmod{n_2}\)
- Composite signature: \(s = \text{CRT}_{n_1,n_2}(s_1, s_2)\)
Security reduction to the RSA

- If RSA is $S$-secure against existential forgeries via adaptive chosen message attack, then the composite signature is about $\frac{S}{t_{ex}}$-secure against the same attack, where $t_{ex}$ is the time for one modular exponentiation.
Security reduction to the RSA

- If RSA is $S$-secure against existential forgeries via adaptive chosen message attack, then the composite signature is about $\frac{S}{t_{ex}}$-secure against the same attack, where $t_{ex}$ is the time for one modular exponentiation.

- $t_{ex}$ shows how much more RSA exponentiation operation is slower than AES encryption operation, experimentally measured to be about $2^{13}$. 

March 13th 2019
Security reduction to the RSA

- If RSA is $S$-secure against existential forgeries via adaptive chosen message attack, then the composite signature is about $\frac{S}{t_{ex}}$-secure against the same attack, where $t_{ex}$ is the time for one modular exponentiation.

- $t_{ex}$ shows how much more RSA exponentiation operation is slower than AES encryption operation, experimentally measured to be about $2^{13}$

<table>
<thead>
<tr>
<th>Security strength (bits)</th>
<th>Symmetric key algorithms</th>
<th>RSA modulus $n$ (bits)</th>
<th>SplitKey composite modulus $n_1n_2$ (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>3TDEA</td>
<td>2048</td>
<td>6144</td>
</tr>
<tr>
<td>128</td>
<td>AES-128</td>
<td>3072</td>
<td>8192</td>
</tr>
<tr>
<td>192</td>
<td>AES-192</td>
<td>7680</td>
<td>16384</td>
</tr>
</tbody>
</table>
eIDAS QSCD/Common Criteria evaluation

- Based on the eIDAS regulation.
- Old Secure Signature Creation Device PP: prEN 14169-2:2012
- Draft Server Signing PP: prEN 419 241-2
eIDAS QSCD/Common Criteria evaluation

- Based on the eIDAS regulation.
- Old Secure Signature Creation Device PP: prEN 14169-2:2012
- Draft Server Signing PP: prEN 419 241-2
- Evaluation lab: TÜViT in Germany
- Consultant lab: CCLabs in Hungary
- Evaluation process started in the beginning of 2017 and finished in the end of 2018
- Evaluation assurance level for server-side component: EAL4 + AVA_VAN.5
- Evaluation assurance level for client-side component: EAL2
Covered threats in eIDAS QSCD

- Signer enrolment: Enrolment Forgery, Random Guessable, PubKey Forgery, MITM
- Signing process: PIN Guessing, Authentication Forgery, Access Control ByPass, Replay, MITM, Cloning, Tampering
- Cryptographic: Signature Forgery, Hash Forgery
- Other: Unauthorized System Access, Audit Log Forgery
Reduced threats, because of applied TC

- Signer enrolment: **Enrolment Forgery**, Random Guessable, PubKey Forgery, MITM
- Signing process: PIN Guessing, **Authentication Forgery**, Access Control ByPass, **Replay**, MITM, Cloning, Tampering
- Cryptographic: Signature Forgery, Hash Forgery
- Other: **Unauthorized System Access**, **Audit Log Forgery**
Policy security requirements for eIDAS QSCD

- Private key: Randomness, Confidentiality, Sole Control to Signer
- Signing process: Hash Integrity
- Cryptographic: Cryptographically Secure Signature Scheme
- Organisational: Qualified Trust Service Provider
Fulfilled requirements, because of applied TC

- Private key: Randomness, **Confidentiality, Sole Control to Signer**
- Signing process: Hash Integrity
- Cryptographic: Cryptographically Secure Signature Scheme
- Organisational: Qualified Trust Service Provider
Smart-ID – A commercial service with SplitKey

- Legally compliant digital signature (eIDAS) and strong authentication service (PSD2) in Europe
- Developed and operated by SK Identity Solutions AS, a private company in Estonia.
Smart-ID – A commercial service with SplitKey

- Legally compliant digital signature (eIDAS) and strong authentication service (PSD2) in Europe
- Developed and operated by SK Identity Solutions AS, a private company in Estonia.
- Used by online banking, retail, telcos, government, etc.
Smart-ID – A commercial service with SplitKey

- Legally compliant digital signature (eIDAS) and strong authentication service (PSD2) in Europe
- Developed and operated by SK Identity Solutions AS, a private company in Estonia.
- Used by online banking, retail, telcos, government, etc.
- SplitKey was originally developed for the Smart-ID service, now spun off to independent product line
Smart-ID authentication flow

- Authentication is started from the RP’s webpage or RP’s app, custom REST API.
- OpenID Connect API supported, but not widely used.
Smart-ID uptake since the launch

- Estonia
- Latvia
- Lithuania
Smart-ID uptake and usage

- 1.88 M active users in total
- 35% of adult population in Estonia, Latvia, and Lithuania
- 43% - 49% of smartphone users
- 30 M transactions per month
Summary

- Successful example of how to use threshold cryptography:
Summary

- Successful example of how to use threshold cryptography:
- to create an efficient digital signature scheme
Summary

- Successful example of how to use threshold cryptography:
  - to create an efficient digital signature scheme
  - ease security evaluation and reduce risks
Summary

- Successful example of how to use threshold cryptography:
  - to create an efficient digital signature scheme
  - ease security evaluation and reduce risks
  - to reach about 2 million users in production within 3 years
Summary

- Successful example of how to use threshold cryptography:
  - to create an efficient digital signature scheme
  - ease security evaluation and reduce risks
  - to reach about 2 million users in production within 3 years

Questions?