Coppercloud: Blind Server-Supported RSA Signatures

26.04.26

Nikita Snetkov Jelizaveta Vakarjuk Alisa Pankova

2002: Identity cards

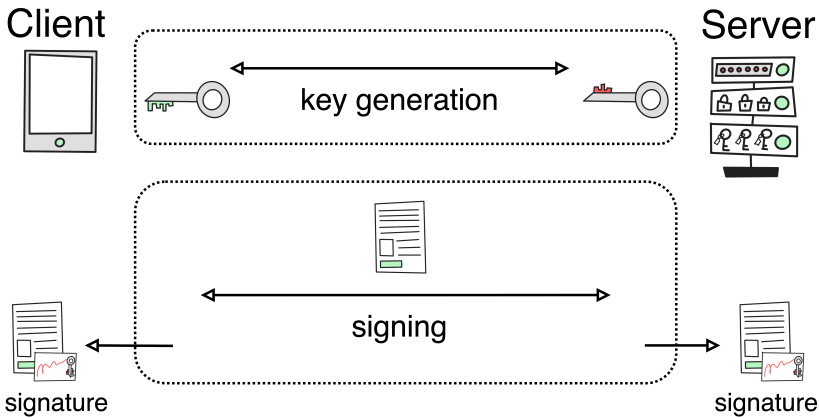
- Create signatures, protecting the private keys
- User's computer must have a card reader
- There are more-or-less standard software components for browsers and other applications to access the card
- To activate the card, it must receive a PIN from the software component or the card reader
  - User enters that PIN

2002: Identity cards

- Create signatures, protecting the private keys
- User's computer must have a card reader
- There are more-or-less standard software components for browsers and other applications to access the card
- To activate the card, it must receive a PIN from the software component or the card reader
  - User enters that PIN

Would it be a good idea to store the private key in a device like smartphone?

2016: SplitKey: the main idea



[1]image author: Jelizaveta Vakarjuk

RSA public key encryption

$$p, q \leftarrow\$ \mathbb{P}$$
$$n \leftarrow p \cdot q$$
Choose an integer e such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$ //public key
$$d \leftarrow e^{-1} \pmod{\phi(n)} \quad \text{//private key}$$

where $\phi(n) = (p-1)(q-1)$ is the Euler's totient function

2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.

2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.

Client

$p_1, q_1 \leftarrow\$ \mathbb{P}$

$n_1 \leftarrow p_1 q_1$

$d_1 \leftarrow e^{-1} \pmod{\phi(n_1)}$

Server

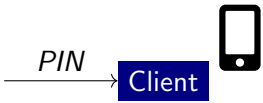
$p_2, q_2 \leftarrow\$ \mathbb{P}$

$n_2 \leftarrow p_2 q_2$

$d_2 \leftarrow e^{-1} \pmod{\phi(n_2)}$

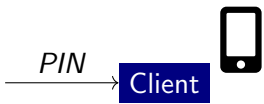
2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.



2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.



$$p_1, q_1 \leftarrow\$ \mathbb{P}$$
$$n_1 \leftarrow p_1 q_1$$
$$d_1 \leftarrow e^{-1} \pmod{\phi(n_1)}$$

$$u \leftarrow\$ \{0,1\}^*$$
$$d_1' \leftarrow \mathrm{GenShare}(PIN, u)$$
$$d_1'' \leftarrow d_1 - d_1' \pmod{\phi(n_1)}$$

$$p_2, q_2 \leftarrow\$ \mathbb{P}$$
$$n_2 \leftarrow p_2 q_2$$
$$d_2 \leftarrow e^{-1} \pmod{\phi(n_2)}$$

2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.

$p_1, q_1 \leftarrow\$ \mathbb{P}$

$n_1 \leftarrow p_1 q_1$

$d_1 \leftarrow e^{-1} \pmod{\phi(n_1)}$

$u \leftarrow\$ \{0,1\}^*$

$d_1' \leftarrow \mathrm{GenShare}(PIN, u)$

$d_1'' \leftarrow d_1 - d_1' \pmod{\phi(n_1)}$

n_1, d_1'' →

← n_2

$p_2, q_2 \leftarrow\$ \mathbb{P}$

$n_2 \leftarrow p_2 q_2$

$d_2 \leftarrow e^{-1} \pmod{\phi(n_2)}$

2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.

Client

$p_1, q_1 \leftarrow\$ \mathbb{P}$

$n_1 \leftarrow p_1 q_1$

$d_1 \leftarrow e^{-1} \pmod{\phi(n_1)}$

$u \leftarrow\$ \{0,1\}^*$

$d_1' \leftarrow \mathrm{GenShare}(PIN, u)$

$d_1'' \leftarrow d_1 - d_1' \pmod{\phi(n_1)}$

$\mathsf{pk} = (n_1 n_2, e)$

n_1, d_1'' ⟶

⟵ n_2

Server

$p_2, q_2 \leftarrow\$ \mathbb{P}$

$n_2 \leftarrow p_2 q_2$

$d_2 \leftarrow e^{-1} \pmod{\phi(n_2)}$

2016: RSA-based SplitKey (key generation)

Let e be an RSA public key.

Client

$p_1, q_1 \leftarrow\$ \mathbb{P}$

$n_1 \leftarrow p_1 q_1$

$d_1 \leftarrow e^{-1} \pmod{\phi(n_1)}$

$u \leftarrow\$ \{0,1\}^*$

$d_1' \leftarrow \mathrm{GenShare}(PIN, u)$

$d_1'' \leftarrow d_1 - d_1' \pmod{\phi(n_1)}$

$\mathsf{pk} = (n_1 n_2, e)$

n_1, d_1'' →

← n_2

Server

$p_2, q_2 \leftarrow\$ \mathbb{P}$

$n_2 \leftarrow p_2 q_2$

$d_2 \leftarrow e^{-1} \pmod{\phi(n_2)}$

2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$

Client

$u,\ n_1,\ n_2$

Server

$d_1'',\ d_2,\ n_1,\ n_2$

2016: RSA-based SplitKey (signing)

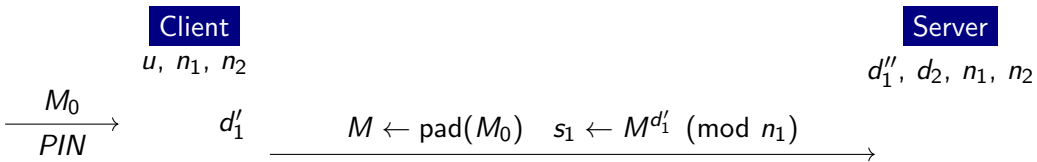
$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$

2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$

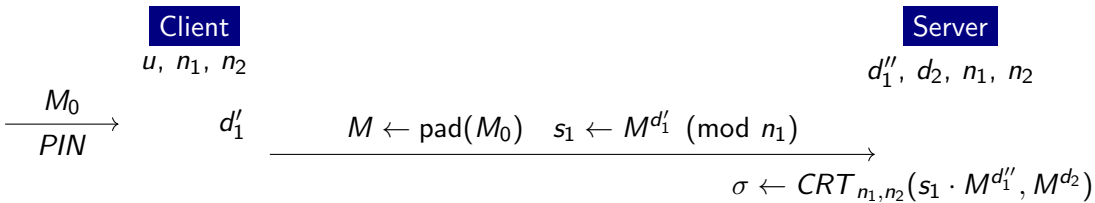
2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$



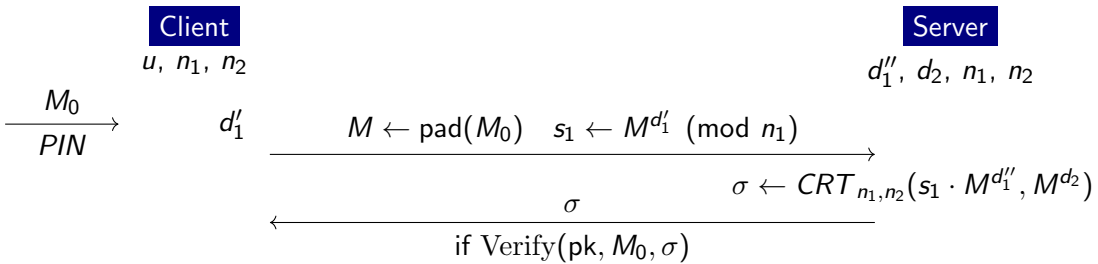
2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$



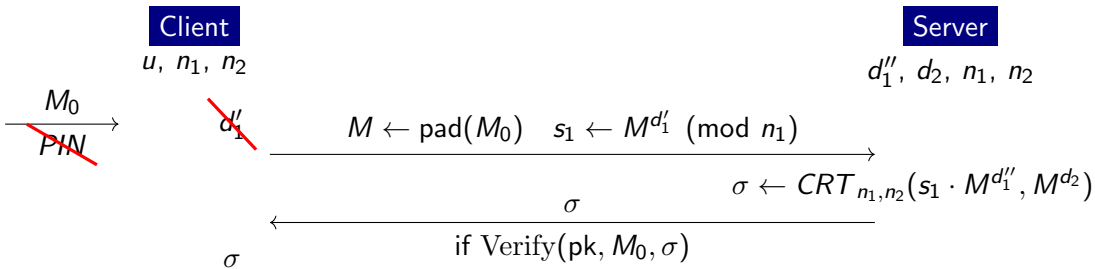
2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$



2016: RSA-based SplitKey (signing)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)} \quad d_1' = \mathrm{GenShare}(PIN, u)$$

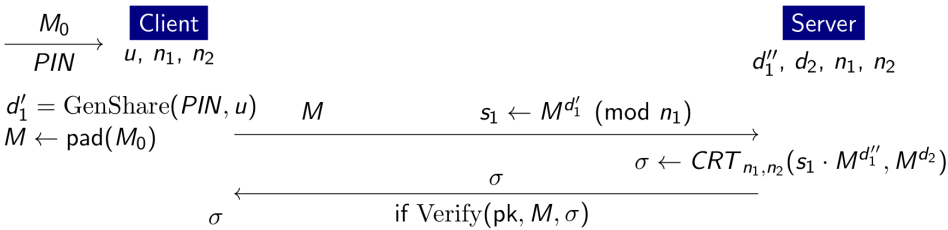


Clone detection

- Additionally, Client and Server share a random bitstring w
- During signing, Client also sends w to Server
- Server checks that the value of w is as expected
  - If not, Server disqualifies this Client / this pk
- Server generates a new w and sends it back, together with σ

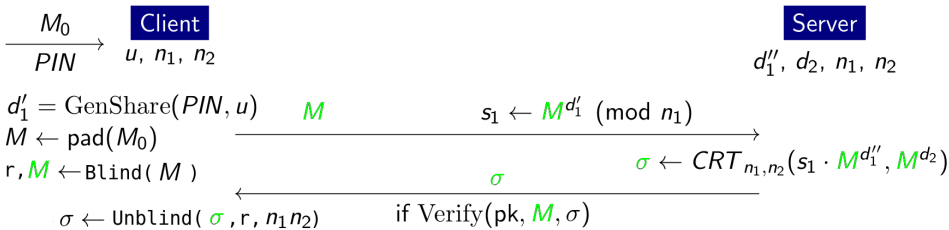
SplitKey signing (without blinding)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}$$

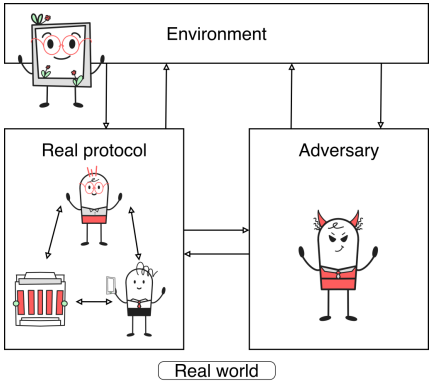


SplitKey signing (with blinding)

$$\mathsf{pk} = (n_1 n_2, e) \quad n_i = p_i q_i \quad d_i \cdot e \equiv 1 \pmod{\varphi(n_i)}$$
$$d_1' + d_1'' \equiv d_1 \pmod{\varphi(n_1)}$$

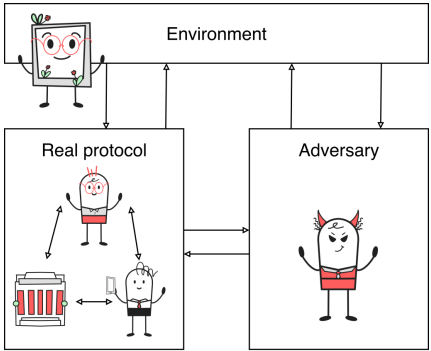


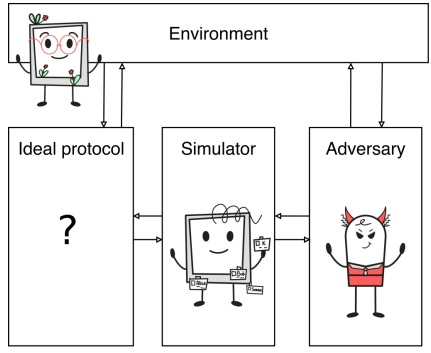
Proofs in the Real-Ideal world paradigm (e.g. Universal Composability)



[2]image author: Jelizaveta Vakarjuk

Proofs in the Real-Ideal world paradigm (e.g. Universal Composability)





How should one define the ideal functionality?

[2]image author: Jelizaveta Vakarjuk

Interactions that need to be simulated (for signing)

- Signing by the Client (honest) and the Server (honest)
- Signing by the Client (honest) and the Adversary – corrupted server case
- Signing by the Adversary and the Server (honest) – corrupted client case

Interactions that need to be simulated (for signing)

- Signing by the Client (honest) and the Server (honest)
  - Signature unforgeability (UF-CMA)
- Signing by the Client (honest) and the Adversary – corrupted server case

- Signing by the Adversary and the Server (honest) – corrupted client case

Interactions that need to be simulated (for signing)

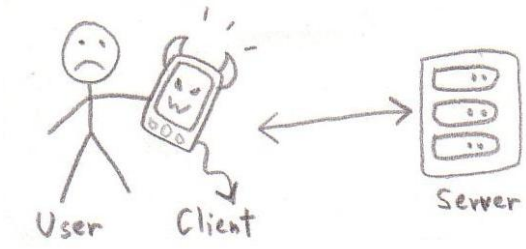
- Signing by the Client (honest) and the Server (honest)
  - Signature unforgeability (UF-CMA)
- Signing by the Client (honest) and the Adversary – corrupted server case
  - The adversary should not tamper with signing (easy to check)
- Signing by the Adversary and the Server (honest) – corrupted client case

Interactions that need to be simulated (for signing)

- Signing by the Client (honest) and the Server (honest)
  - Signature unforgeability (UF-CMA)
- Signing by the Client (honest) and the Adversary – corrupted server case
  - The adversary should not tamper with signing (easy to check)
  - The adversary should not learn the message to be signed (blinding)
- Signing by the Adversary and the Server (honest) – corrupted client case

Interactions that need to be simulated (for signing)

- Signing by the Client (honest) and the Server (honest)
  - Signature unforgeability (UF-CMA)
- Signing by the Client (honest) and the Adversary – corrupted server case
  - The adversary should not tamper with signing (easy to check)
  - The adversary should not learn the message to be signed (blinding)
- Signing by the Adversary and the Server (honest) – corrupted client case
  - What a corrupted client should be able to do?



Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".

Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request

Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request
- If Adversary fully takes over Client, then it can sign whatever it likes.

Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request
- If Adversary fully takes over Client, then it can sign whatever it likes.
  - We can give up on protecting the client in this case.

Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request
- If Adversary fully takes over Client, then it can sign whatever it likes.
  - We can give up on protecting the client in this case.
  - ... or, we can limit the number of signing sessions, or allow the client to contact the server, still protecting the user in the case of corrupted device.

Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request
- If Adversary fully takes over Client, then it can sign whatever it likes.
  - We can give up on protecting the client in this case.
  - ... or, we can limit the number of signing sessions, or allow the client to contact the server, still protecting the user in the case of corrupted device.
    - Non-blinded signing: a signature σ on m is generated only if approved by the server.

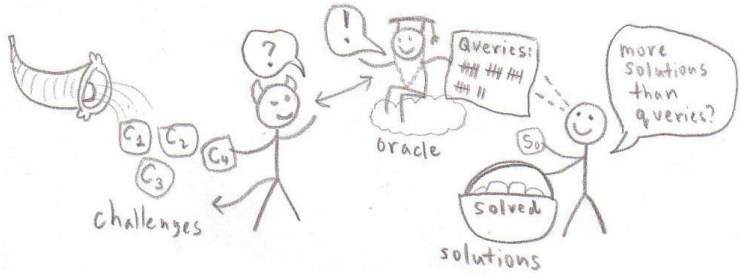
Possible client corruptions that we want to model for SplitKey

- Leak of client's encrypted memory (u as encryption of d_1'): u, n_1, n_2
  - Adversary tries to brute-force *PIN* and find d_1'.
  - Adversary cannot verify its guess without contacting Server.
  - Server maintains "wrong PIN counter".
- Leak of client's unencrypted memory: d_1', u, n_1, n_2
  - Adversary can issue signing requests until Client also issues a request
- If Adversary fully takes over Client, then it can sign whatever it likes.
  - We can give up on protecting the client in this case.
  - ... or, we can limit the number of signing sessions, or allow the client to contact the server, still protecting the user in the case of corrupted device.
    - Non-blinded signing: a signature σ on m is generated only if approved by the server.
    - Blinded signing: the client should not be able to sign more messages than approved by the server.

Proving a limit on the number of signatures a client can generate

One-more cryptographic assumptions:

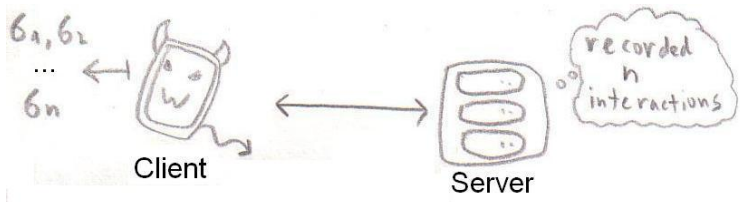
- The adversary is given many instances of a cryptographic challenge (e.g. to solve the RSA problem).
- The adversary can query an oracle $O(\cdot)$ that allows to solve the RSA probem (on any input of adversary's choice).
- The adversary wins if it solves $n+1$ different challenges while quering $O(\cdot)$ at most n times.



Proving a limit on the number of signatures a client can generate

RSA-ACTI (Alternative Chosen-Target RSA Inversion) problem: [Bellare, 2002].

- Intuitively, getting n different RSA signatures from the server does not allow the adversary to somehow obtain one more signature.
- The adversary gets as many signatures as there have been signing sessions.
- Under RSA-ACTI assuptions, the client should not be able to sign more messages than approved by the server.



So what we have added to SplitKey?

- The (un)blinding can be performed locally by the client, without the need to modify server's computation, or the signature verification.
- The ideal functionality needed more modifications than the real protocol.
- As RSA-ACTI is a stronger assumption than RSA, has SplitKey become weaker???

So what we have added to SplitKey?

- The (un)blinding can be performed locally by the client, without the need to modify server's computation, or the signature verification.
- The ideal functionality needed more modifications than the real protocol.
- As RSA-ACTI is a stronger assumption than RSA, has SplitKey become weaker???
  - Non-blinded signing: a signature σ on m is generated only if approved by the server.
  - Blinded signing: the client should not be able to sign more messages than approved by the server.

So what we have added to SplitKey?

- The (un)blinding can be performed locally by the client, without the need to modify server's computation, or the signature verification.
- The ideal functionality needed more modifications than the real protocol.
- As RSA-ACTI is a stronger assumption than RSA, has SplitKey become weaker???
  - Non-blinded signing: a signature σ on m is generated only if approved by the server.
  - Blinded signing: the client should not be able to sign more messages than approved by the server.
- ...the non-blinded version does not guarantee that the adversary cannot somehow magically obtain one more signature.
- In practice, the adversary would need one more sensible signature, which can be much harder.

Thank You!