

RSA CONFERENCE 2014

FEBRUARY 24 - 28 | MOSCONE CENTER | SAN FRANCISCO

Share.
Learn.
Secure.

Capitalizing on
Collective Intelligence

Rethinking Verifiably Encrypted Signatures: A Gap in Functionality and Potential Solutions

SESSION ID: ·cryp-r02

Sarah Meiklejohn

Graduate researcher
UC San Diego



Models for cryptographic primitives



Models for cryptographic primitives

- ◆ In cryptography, we put a lot of effort into accurately modeling **security**: what an adversary can and can't do



Models for cryptographic primitives

- ◆ In cryptography, we put a lot of effort into accurately modeling **security**: what an adversary can and can't do
- ◆ But it's also incredibly important to accurately model **functionality**!



Models for cryptographic primitives

- ◆ In cryptography, we put a lot of effort into accurately modeling **security**: what an adversary can and can't do
- ◆ But it's also incredibly important to accurately model **functionality**!
- ◆ We look at definitions for verifiably encrypted signatures (VES)



Models for cryptographic primitives

- ◆ In cryptography, we put a lot of effort into accurately modeling **security**: what an adversary can and can't do
- ◆ But it's also incredibly important to accurately model **functionality**!
- ◆ We look at definitions for verifiably encrypted signatures (VES)
 - ◆ First show a generic construction based **solely on signatures**



Models for cryptographic primitives

- ◆ In cryptography, we put a lot of effort into accurately modeling **security**: what an adversary can and can't do
- ◆ But it's also incredibly important to accurately model **functionality**!
- ◆ We look at definitions for verifiably encrypted signatures (VES)
 - ◆ First show a generic construction based **solely on signatures**
 - ◆ Then propose **new definition(s)**



An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



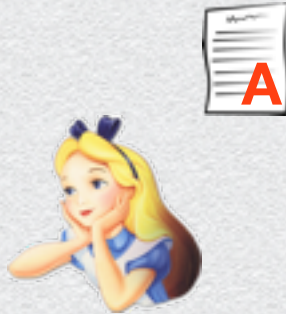
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



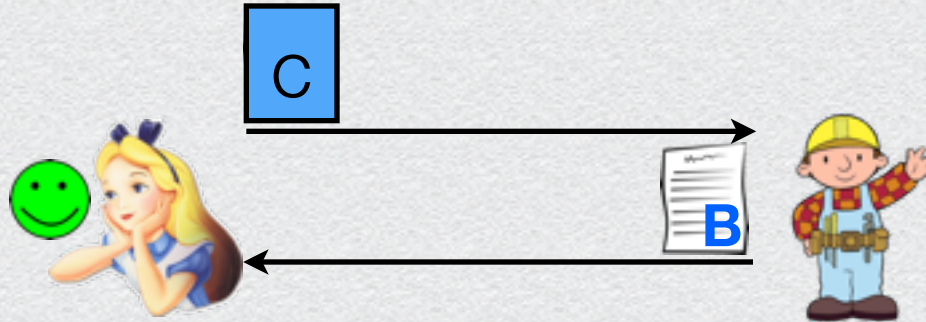
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



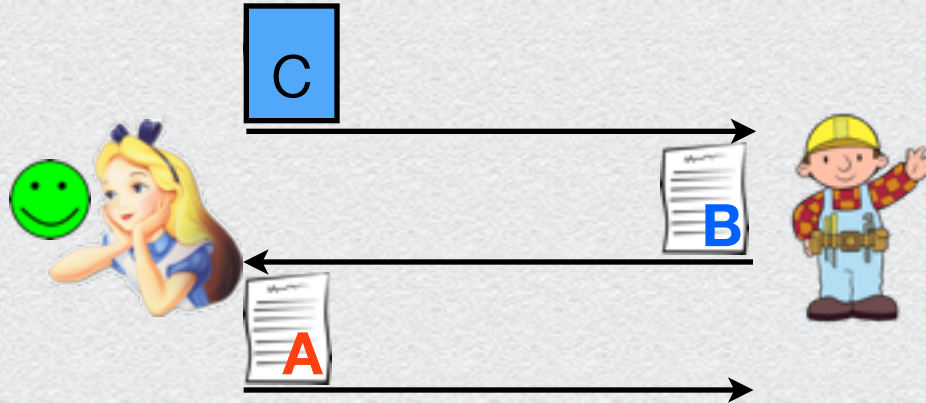
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



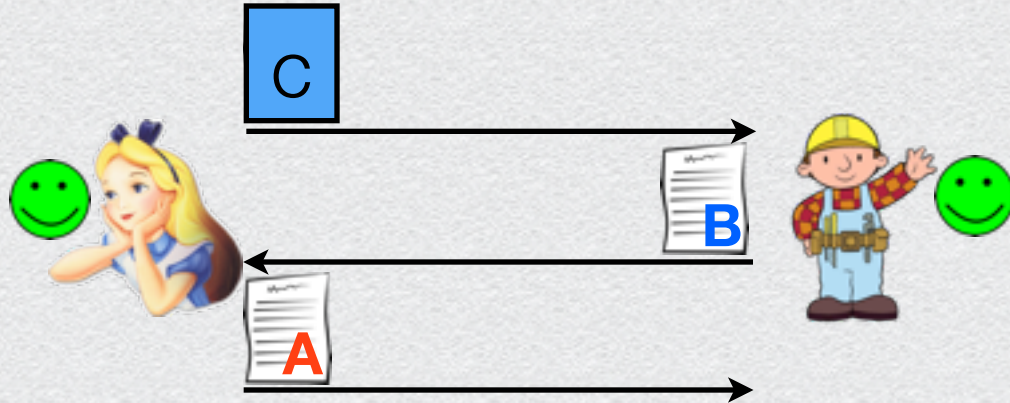
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



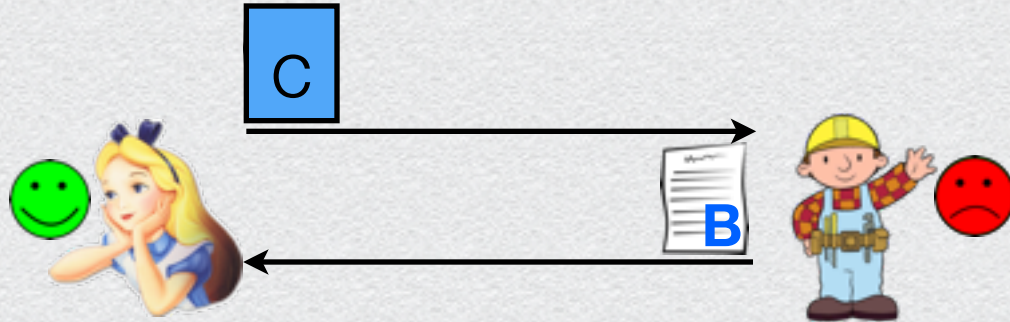
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



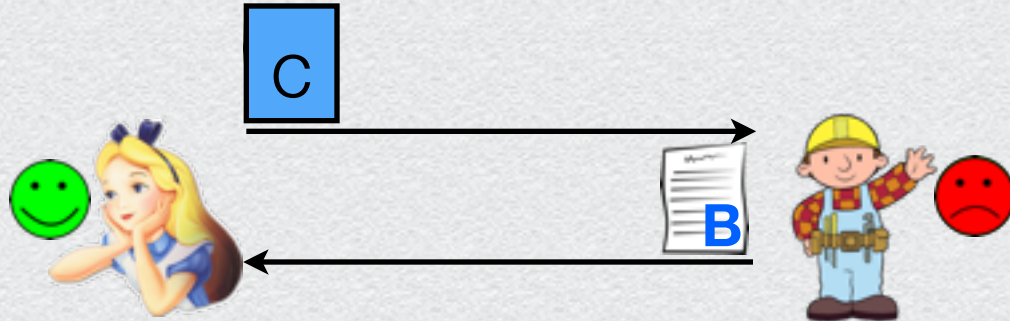
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



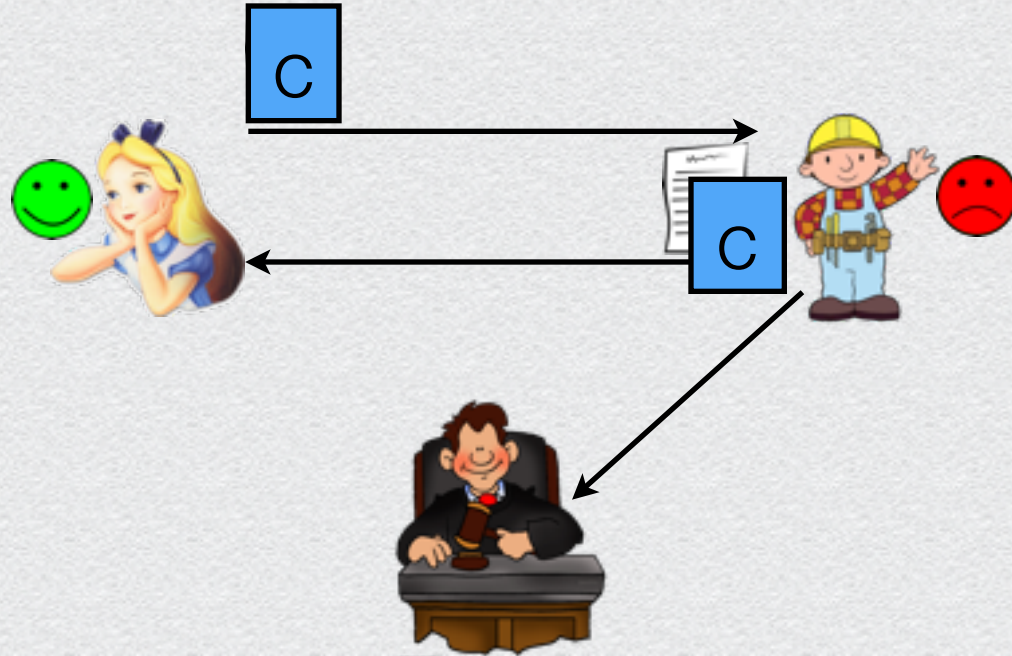
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



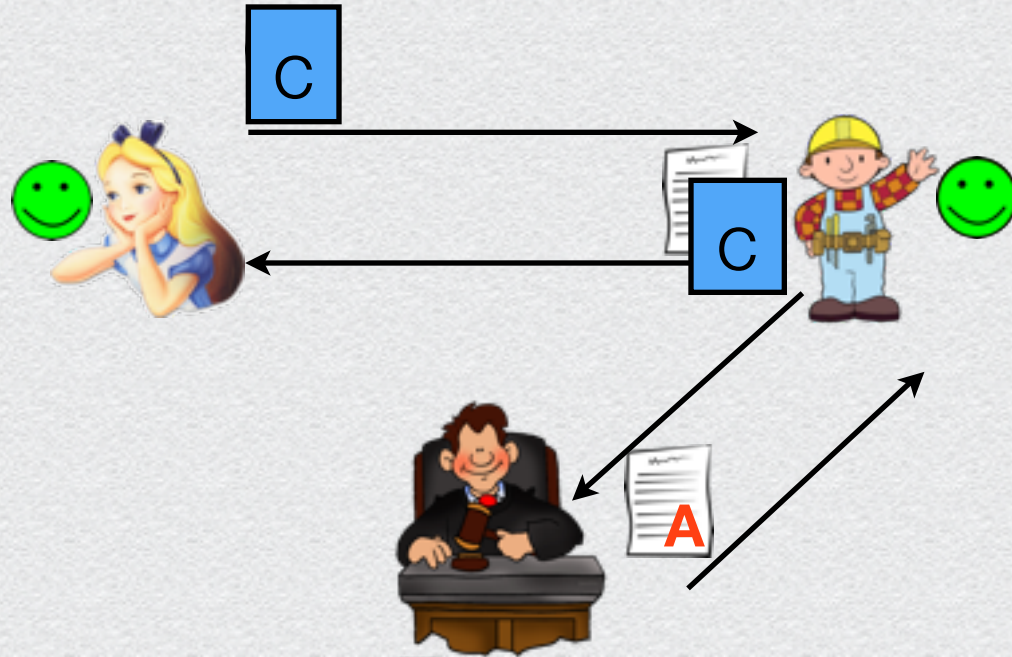
An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



An application: fair contract signing [ASW88]

- ◆ Alice and Bob want each other's signature on a particular document, but they don't trust each other



RSACONFERENCE2014

FEBRUARY 24 – 28 | MOSCONE CENTER | SAN FRANCISCO



Definitions for VES:

- ◆ **Unforgeability**
- ◆ **Opacity**
- ◆ **Extractability**

Verifiably encrypted signatures [BGLS03]



Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, AKG, \text{VESign}, \text{VEVerify}, \text{Resolve})$



Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple (KG, Sign, Verify, AKG, VESign, VESVerify, Resolve)



Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



$(pk_A, sk_A) \leftarrow KG(1^k)$



$(pk_B, sk_B) \leftarrow KG(1^k)$

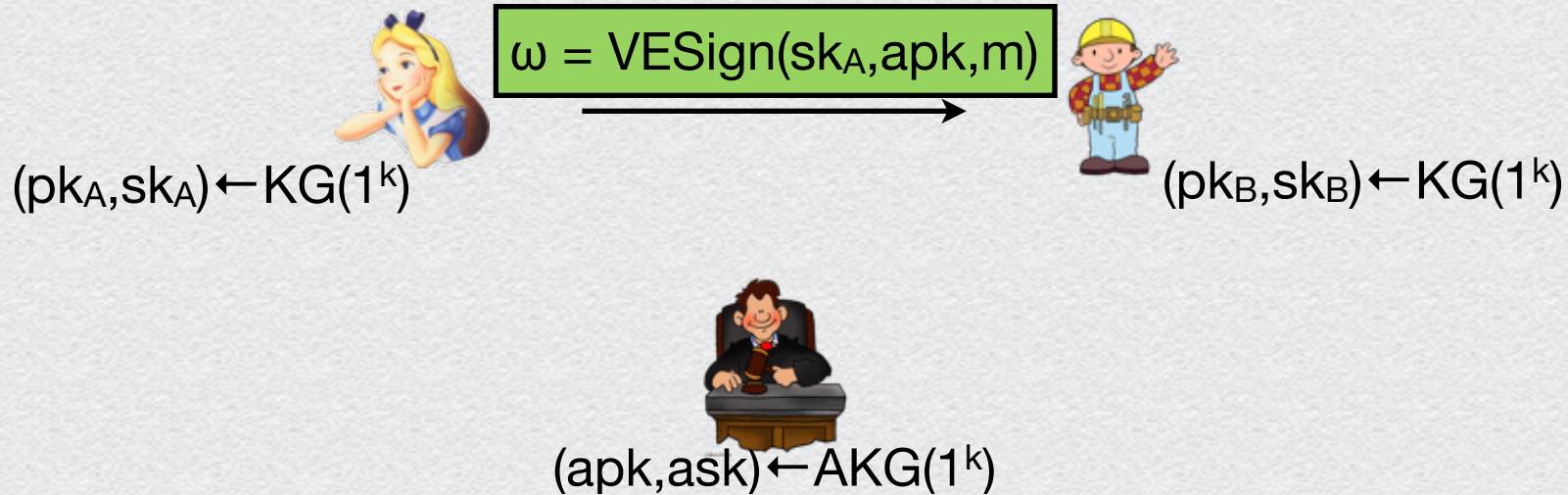


$(apk, ask) \leftarrow AKG(1^k)$



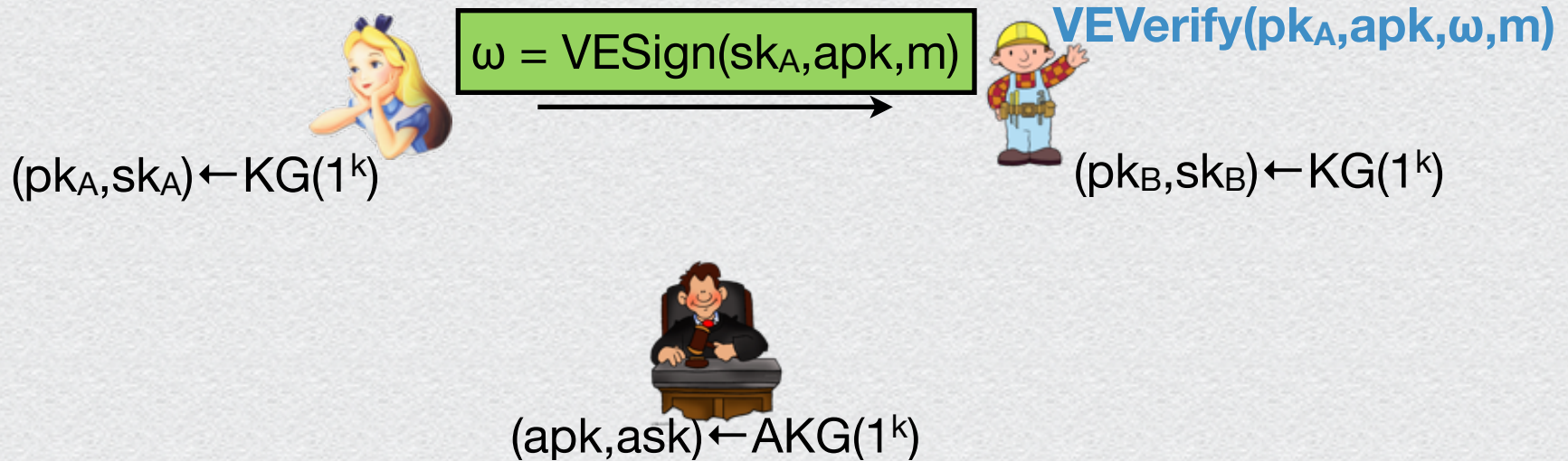
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



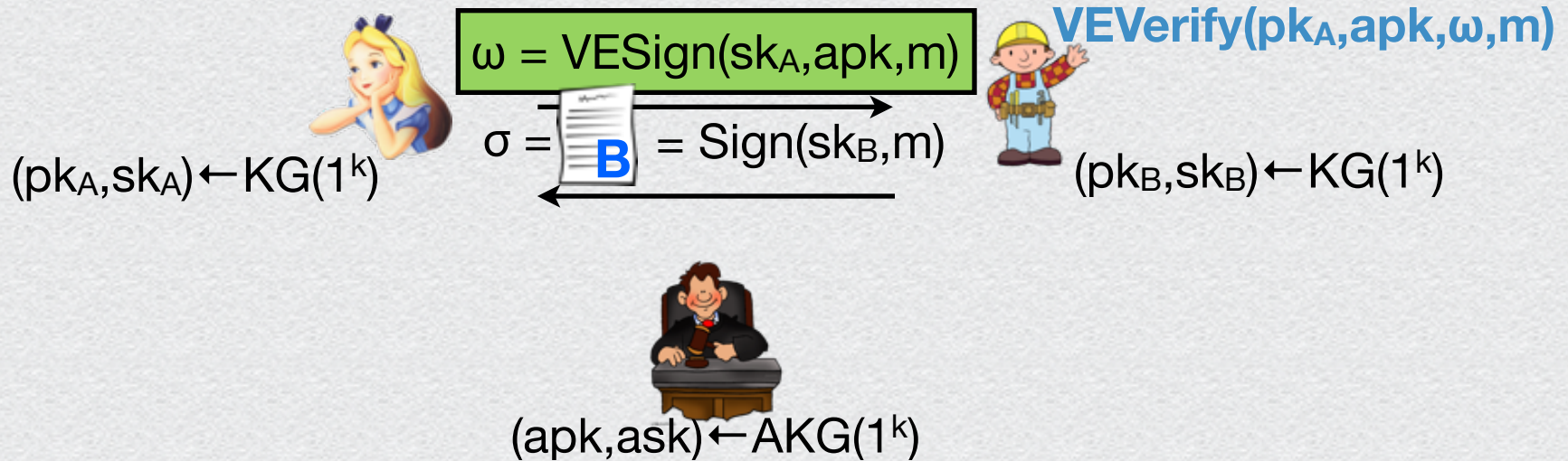
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



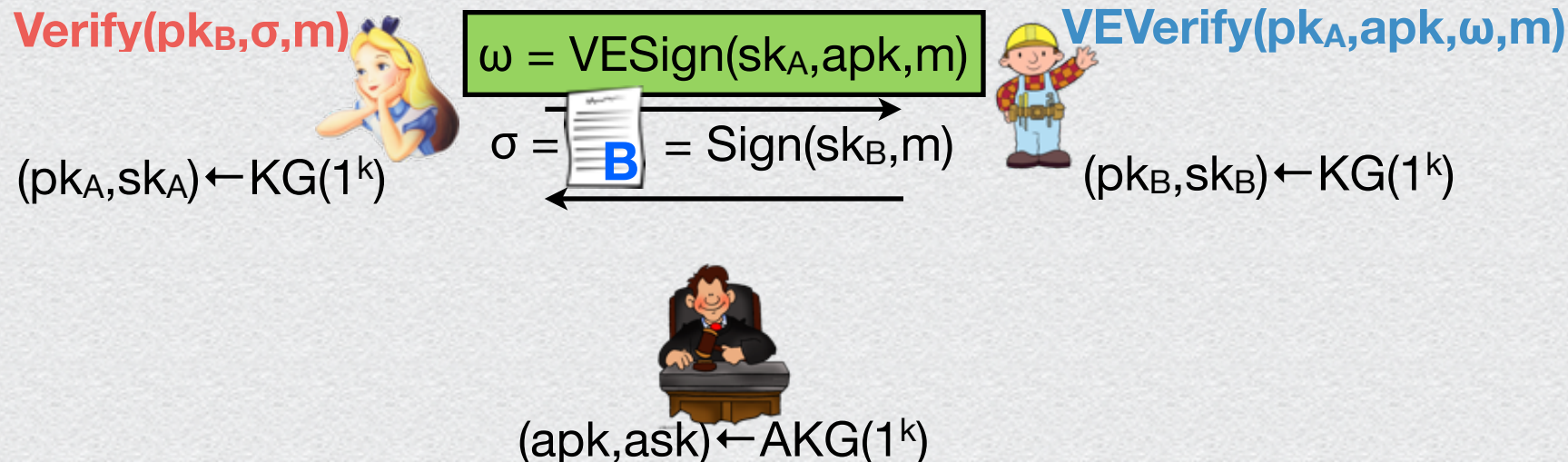
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



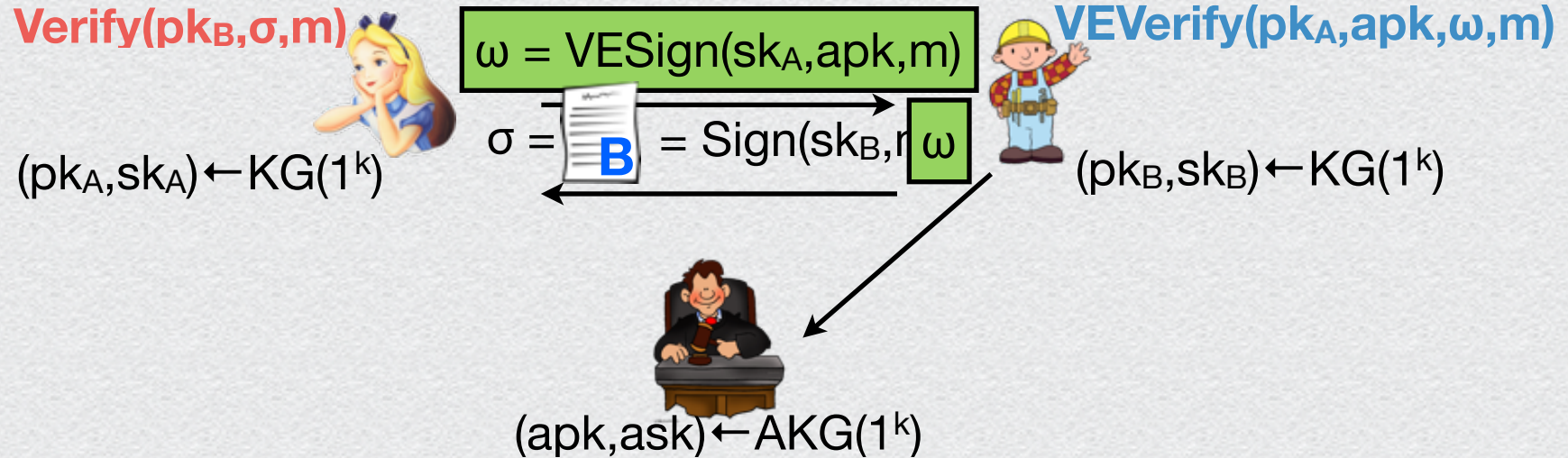
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



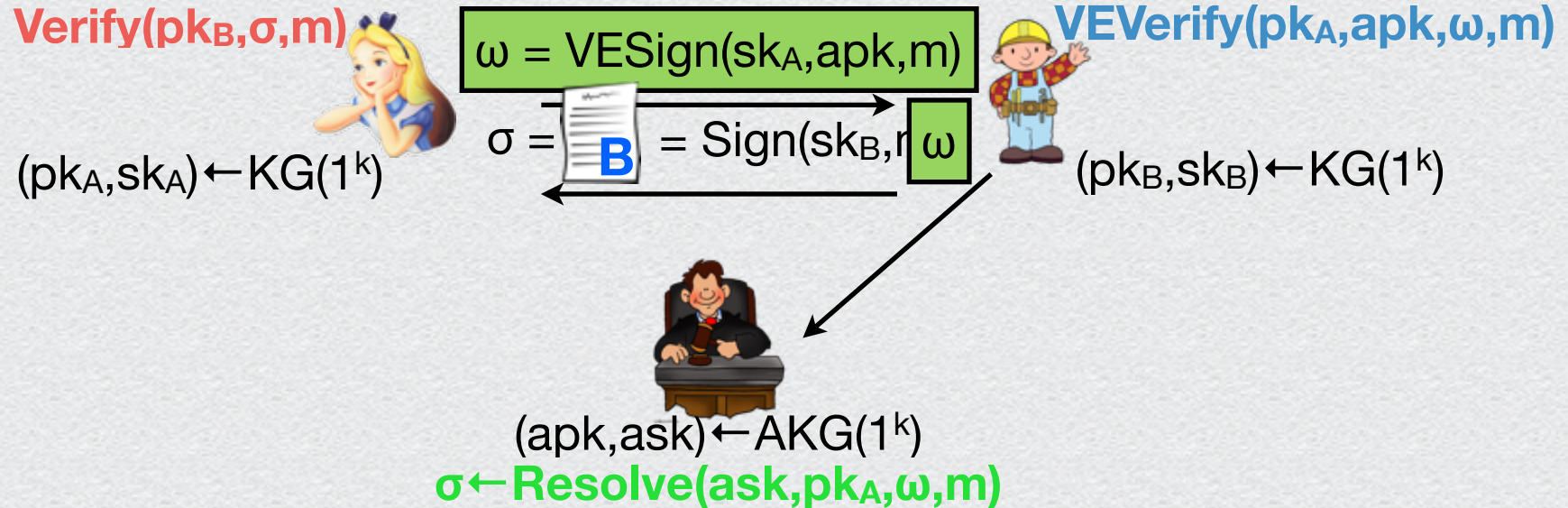
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



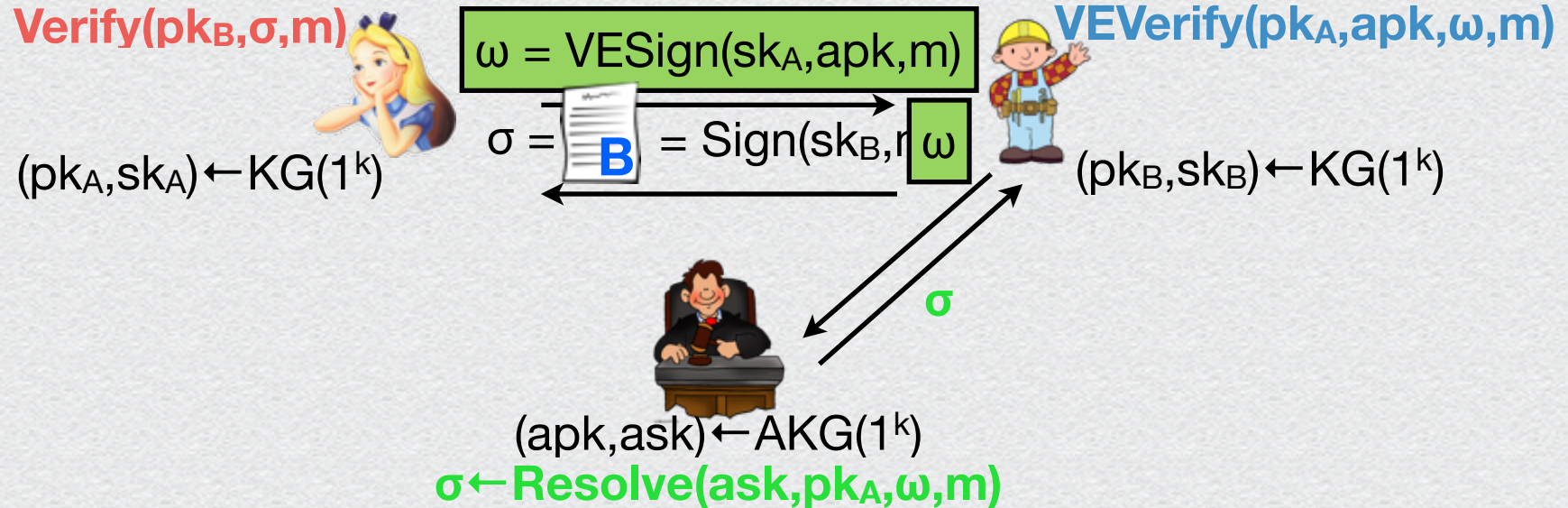
Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



Verifiably encrypted signatures [BGLS03]

- ◆ A VES is a tuple $(KG, \text{Sign}, \text{Verify}, \text{AKG}, \text{VESign}, \text{VEVerify}, \text{Resolve})$



Verifiably encrypted signatures [BGLS03]



Verifiably encrypted signatures [BGLS03]

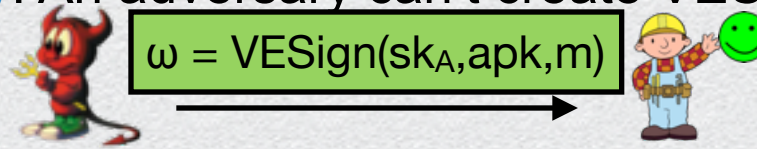
- ◆ A **secure** VES satisfies three properties:



Verifiably encrypted signatures [BGLS03]

- ◆ A **secure** VES satisfies three properties:

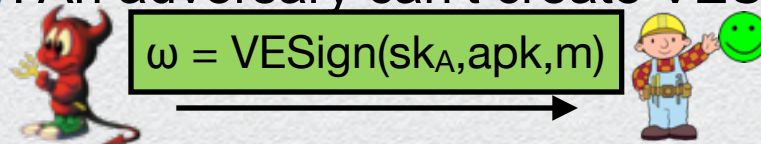
- ◆ **Unforgeability**: An adversary can't create VES



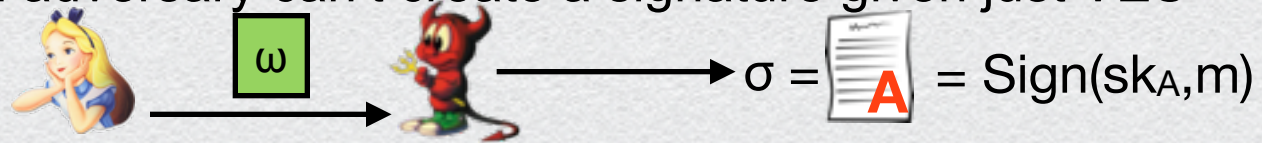
Verifiably encrypted signatures [BGLS03]

- ◆ A **secure** VES satisfies three properties:

- ◆ **Unforgeability**: An adversary can't create VES



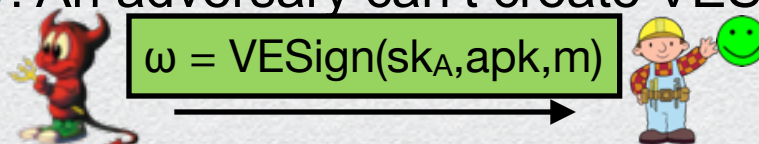
- ◆ **Opacity**: An adversary can't create a signature given just VES



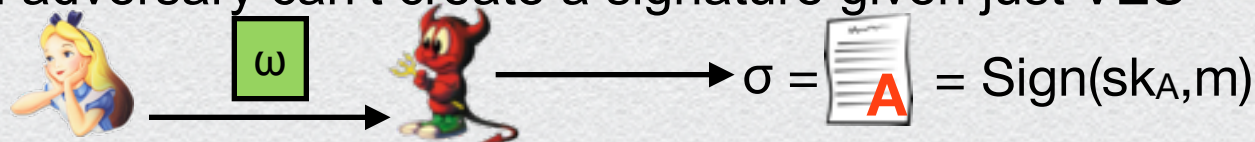
Verifiably encrypted signatures [BGLS03]

- ◆ A **secure** VES satisfies three properties:

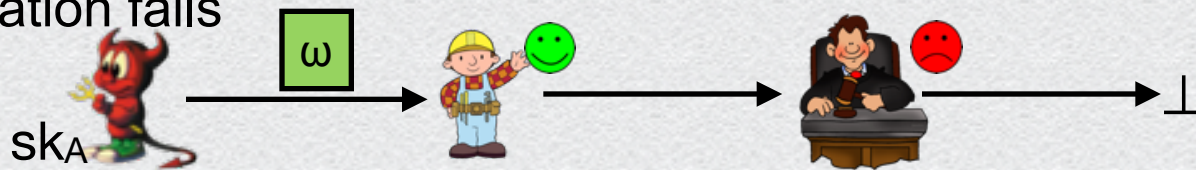
- ◆ **Unforgeability**: An adversary can't create VES



- ◆ **Opacity**: An adversary can't create a signature given just VES



- ◆ **Extractability**: An adversary can't create valid VES for which arbitration fails



RSA CONFERENCE 2014

FEBRUARY 24 - 28 | MOSCONE CENTER | SAN FRANCISCO



A signature-based VES

Constructing VES with just signatures



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign , VESign , and Resolve all use Sign' , just sign different messages



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign, VESign, and Resolve all use Sign' , just sign different messages

Sign

VESign

Resolve



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign , VESign , and Resolve all use Sign' , just sign different messages

Sign



$T(m, \perp, \perp, \perp)$

VESign

Resolve



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign , VESign , and Resolve all use Sign' , just sign different messages

Sign



$T(m, \perp, \perp, \perp)$

VESign



$T(m, \text{apk}, 0, \perp)$

Resolve



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign , VESign , and Resolve all use Sign' , just sign different messages

Sign



$T(m, \perp, \perp, \perp)$

VESign



$T(m, \text{apk}, 0, \perp)$

Resolve



$T(m, \text{apk}, 1, \omega)$



Constructing VES with just signatures

- ◆ Assume we have a signature $(KG', \text{Sign}', \text{Verify}')$ with message space M' and a transformation T from $(M, \text{APK}, 0/1, \Omega)$ to M'
- ◆ Sign , VESign , and Resolve all use Sign' , just sign different messages

Sign



$T(m, \perp, \perp, \perp)$

VESign



$T(m, \text{apk}, 0, \perp)$

Resolve



$T(m, \text{apk}, 1, \omega)$

- ◆ There are **two signatures**, and Verify checks for both



Security of signature-based construction



Security of signature-based construction

- ◆ Unforgeability: can't create VES



Security of signature-based construction

- ◆ Unforgeability: can't create VES



$T(m, apk, 0, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES



$T(m, apk, 0, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES



$T(m, apk, 0, \perp)$

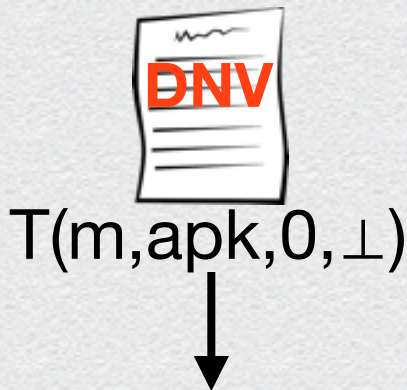
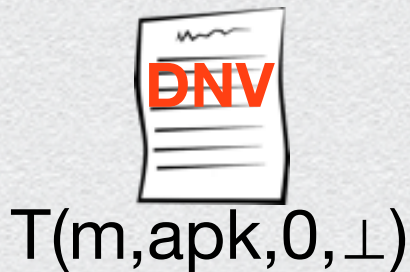


$T(m, apk, 0, \perp)$




Security of signature-based construction




- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES


 $T(m, apk, 0, \perp)$


 $T(m, apk, 0, \perp)$


 $T(m, \perp, \perp, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES
- ◆ Extractability: can't create VES for which arbitration fails



$T(m, apk, 0, \perp)$



$T(m, apk, 0, \perp)$



$T(m, \perp, \perp, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES
- ◆ Extractability: can't create VES for which arbitration fails



$T(m, apk, 0, \perp)$



$T(m, apk, 0, \perp)$



$T(m, \perp, \perp, \perp)$



$T(m, apk, 0, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES
- ◆ Extractability: can't create VES for which arbitration fails



$T(m, apk, 0, \perp)$



$T(m, apk, 0, \perp)$



$T(m, \perp, \perp, \perp)$



$T(m, apk, 0, \perp)$



Security of signature-based construction

- ◆ Unforgeability: can't create VES
- ◆ Opacity: can't create signature given VES
- ◆ Extractability: can't create VES for which arbitration fails



$T(m, apk, 0, \perp)$



$T(m, apk, 0, \perp)$



$T(m, \perp, \perp, \perp)$



$T(m, apk, 0, \perp)$



$T(m, apk, 1, \omega)$



RSA CONFERENCE 2014

FEBRUARY 24 - 28 | MOSCONE CENTER | SAN FRANCISCO



**Resolution
independence**

Resolution independence

- ◆ The problem with the signature-based construction: Bob got a different object from Alice than from the arbiter!



Resolution independence

- ◆ The problem with the signature-based construction: Bob got a different object from Alice than from the arbiter!



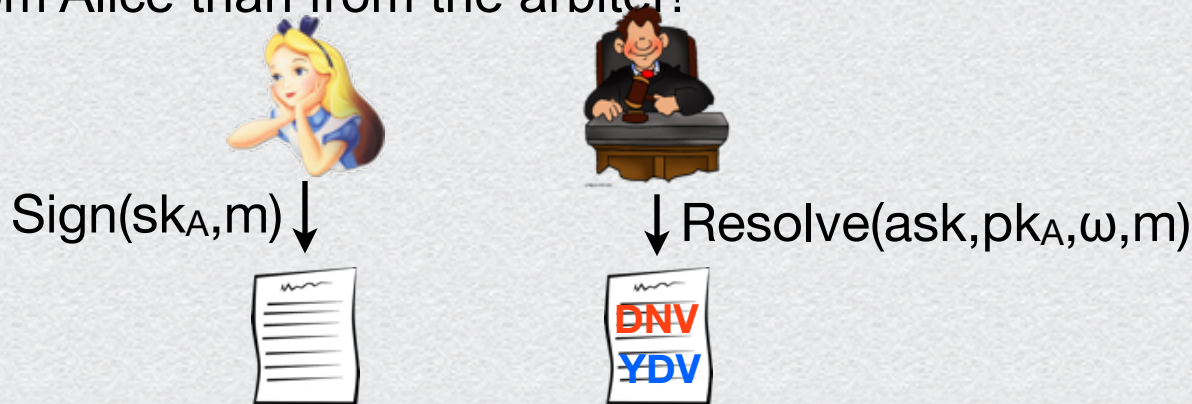
Resolution independence

- ◆ The problem with the signature-based construction: Bob got a different object from Alice than from the arbiter!



Resolution independence

- ◆ The problem with the signature-based construction: Bob got a different object from Alice than from the arbiter!



- ◆ **Resolution independence:** the distributions $\{\text{Sign}(sk, m)\}$ and $\{\text{Resolve}(ask, pk, \omega, m)\}$ are identical



Separating our construction from existing ones



Separating our construction from existing ones

- ◆ Signature construction is not resolution independent: σ vs. (apk, ω, ω')



Separating our construction from existing ones

- ◆ Signature construction is not resolution independent: σ vs. (apk, ω, ω')
- ◆ But it is satisfied by all existing VES constructions
 - ◆ [BGLS03] uses bilinear groups, BLS signatures, deterministic Resolve
 - ◆ [LOSSW05] uses bilinear groups, Waters signatures, randomized Resolve
 - ◆ [R09] uses RSA groups and signatures, deterministic Resolve



Resolution duplication



Resolution duplication

- ◆ Verifiably **encrypted** signatures: encryption really must be happening



Resolution duplication

- ◆ Verifiably **encrypted** signatures: encryption really must be happening
- ◆ Can form ω so that no one can extract σ from ω (by **opacity**), except the arbiter can extract σ' from the same distribution (by **resolution independence**)



Resolution duplication

- ◆ Verifiably **encrypted** signatures: encryption really must be happening
- ◆ Can form ω so that no one can extract σ from ω (by **opacity**), except the arbiter can extract σ' from the same distribution (by **resolution independence**)
- ◆ Not quite encryption: σ' might be different from σ



Resolution duplication

- ◆ Verifiably **encrypted** signatures: encryption really must be happening
- ◆ Can form ω so that no one can extract σ from ω (by **opacity**), except the arbiter can extract σ' from the same distribution (by **resolution independence**)
- ◆ Not quite encryption: σ' might be different from σ

- ◆ **Resolution duplication** requires: (1) resolution independence, (2) deterministic Resolve, and (3) that there exists an algorithm Extract such that $\text{Extract}(\text{sk}, m, r) = \text{Resolve}(\text{ask}, \text{pk}, \text{VESign}(\text{sk}, \text{apk}, m; r), m)$



Constructing PKE with resolution duplication



Constructing PKE with resolution duplication

- ◆ With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$




Constructing PKE with resolution duplication

- ◆ With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$



Constructing PKE with resolution duplication

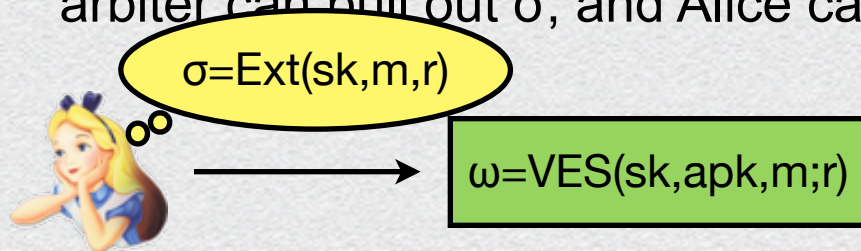
- ◆ With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$


$$\sigma = \text{Ext}(\text{sk}, m, r)$$



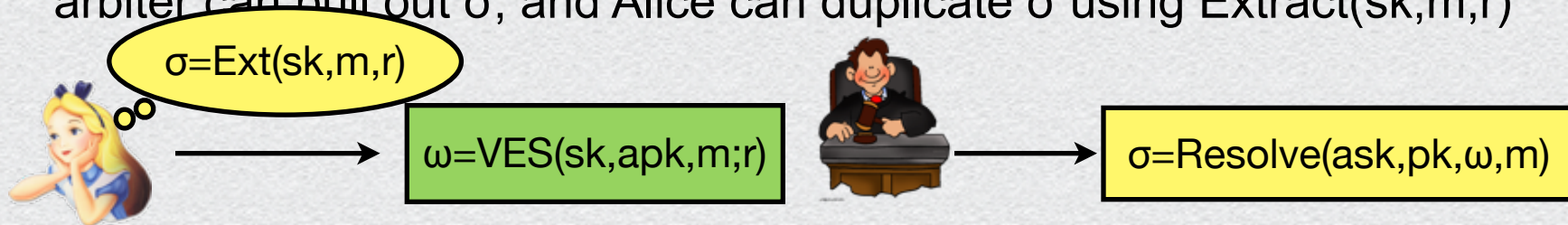
Constructing PKE with resolution duplication

- ◆ With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$



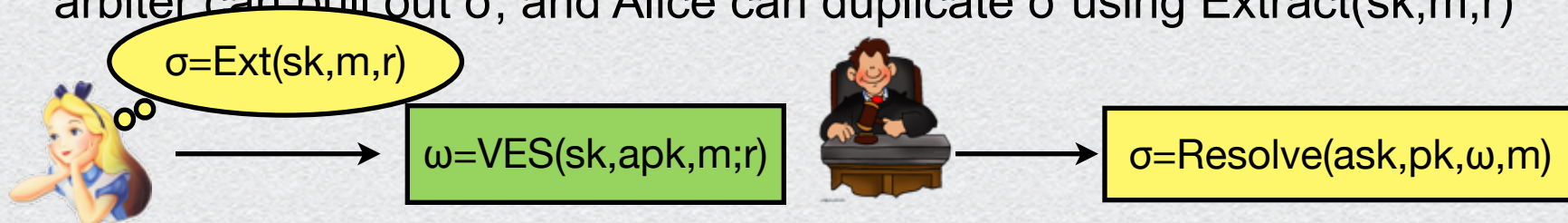
Constructing PKE with resolution duplication

- With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$



Constructing PKE with resolution duplication

- With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$

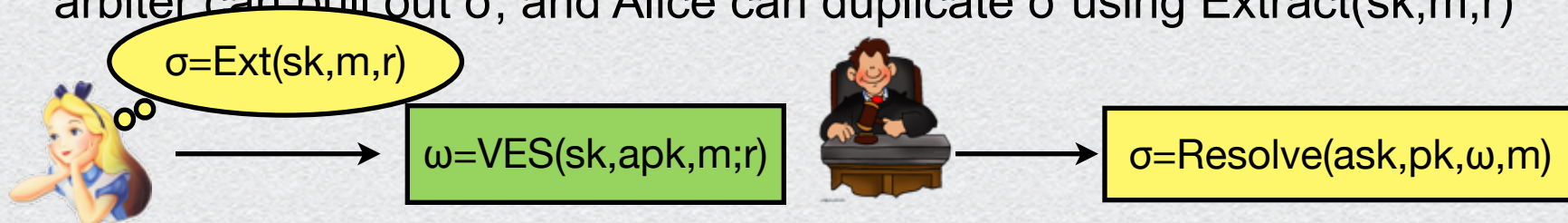


- This lets us “encrypt” signatures, but we want to encrypt arbitrary bits



Constructing PKE with resolution duplication

- With resolution duplication, Alice can form $\omega := \text{VESign}(\text{sk}, \text{apk}, m; r)$ so that no one can pull out $\text{Sign}(\text{sk}, m)$ from ω (by opacity), except the arbiter can pull out σ , and Alice can duplicate σ using $\text{Extract}(\text{sk}, m, r)$




- This lets us “encrypt” signatures, but we want to encrypt arbitrary bits
- Adapt Goldreich-Levin trick [GL89]; show that it is hard to predict (compute) $\langle \sigma, r \rangle = \sum \sigma_i \cdot r_i \bmod 2$ given just ω and r



Constructing PKE with resolution duplication




Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 





Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$




Constructing PKE with resolution duplication

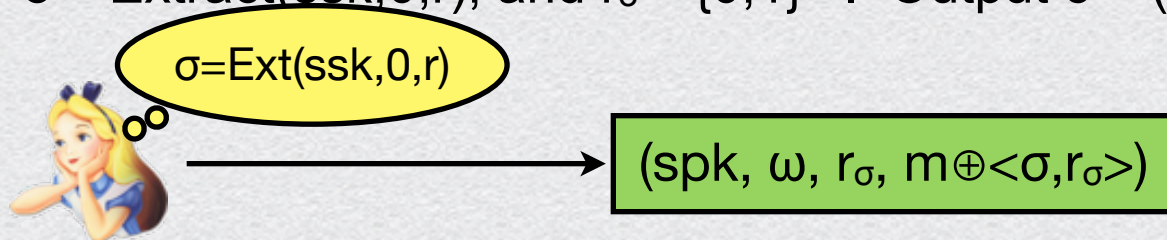
- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$


$$\sigma = \text{Ext}(ssk, 0, r)$$




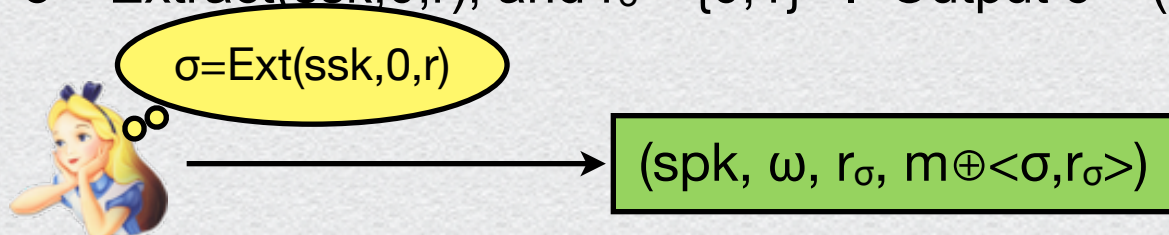
Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$



Constructing PKE with resolution duplication


- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$





- ◆ Dec(sk, c): Parse $c = (c_1, c_2, c_3, c_4)$. Compute $\sigma = \text{Resolve}(sk, c_1, c_2, 0)$ and output $c_4 \oplus \langle \sigma, c_3 \rangle$



Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$



$$\sigma = \text{Ext}(ssk, 0, r)$$



$$(spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$$

- ◆ Dec(sk, c): Parse $c = (c_1, c_2, c_3, c_4)$. Compute $\sigma = \text{Resolve}(sk, c_1, c_2, 0)$ and output $c_4 \oplus \langle \sigma, c_3 \rangle$


$$\sigma = \text{Resolve}(ask, pk, \omega, 0)$$


Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$


$$\sigma = \text{Ext}(ssk, 0, r)$$


$$(spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$$


- ◆ Dec(sk, c): Parse $c = (c_1, c_2, c_3, c_4)$. Compute $\sigma = \text{Resolve}(sk, c_1, c_2, 0)$ and output $c_4 \oplus \langle \sigma, c_3 \rangle$


$$\sigma = \text{Resolve}(ask, pk, \omega, 0)$$

$$c_4 \oplus \langle \sigma, c_3 \rangle = m \oplus \langle \sigma, r_\sigma \rangle \oplus \langle \sigma, c_3 \rangle = m$$

Constructing PKE with resolution duplication

- ◆ EKeyGen(1^k): Output $(pk, sk) \leftarrow \text{AKG}(1^k)$ 
- ◆ Enc(pk, m): Generate $(spk, ssk) \leftarrow \text{KG}(1^k)$, $\omega \leftarrow \text{VESign}(ssk, pk, 0; r)$, $\sigma \leftarrow \text{Extract}(ssk, 0, r)$, and $r_\sigma \leftarrow \{0, 1\}^{|\sigma|}$. Output $c = (spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$


$$\sigma = \text{Ext}(ssk, 0, r)$$

$$(spk, \omega, r_\sigma, m \oplus \langle \sigma, r_\sigma \rangle)$$

The same by
resolution duplication!

- ◆ Dec(sk, c): Parse $c = (c_1, c_2, c_3, c_4)$. Compute $\sigma = \text{Resolve}(sk, c_1, c_2, 0)$ and output $c_4 \oplus \langle \sigma, c_3 \rangle$


$$\sigma = \text{Resolve}(ask, pk, \omega, 0)$$

$$c_4 \oplus \langle \sigma, c_3 \rangle = m \oplus \langle \sigma, r_\sigma \rangle \oplus \langle \sigma, c_3 \rangle = m$$

Constructing PKE with resolution duplication



Constructing PKE with resolution duplication

- ◆ Interestingly, resolution duplication contributed to the **correctness** of the encryption scheme rather than its security

$$c_4 \oplus \langle \sigma, c_3 \rangle = m \oplus \langle \sigma, r_\sigma \rangle \oplus \langle \sigma, c_3 \rangle = m$$



Constructing PKE with resolution duplication

- ◆ Interestingly, resolution duplication contributed to the **correctness** of the encryption scheme rather than its security

$$c_4 \oplus \langle \sigma, c_3 \rangle = m \oplus \langle \sigma, r_\sigma \rangle \oplus \langle \sigma, c_3 \rangle = m$$

- ◆ **IND-CPA security** follows fairly directly from opacity



RSA CONFERENCE 2014

FEBRUARY 24 - 28 | MOSCONE CENTER | SAN FRANCISCO



Conclusions

Conclusions and open problems



Conclusions and open problems

- ◆ Existing VES definitions might not capture desired functionality



Conclusions and open problems

- ◆ Existing VES definitions might not capture desired functionality
 - ◆ Provided a solely signature-based VES



Conclusions and open problems

- ◆ Existing VES definitions might not capture desired functionality
 - ◆ Provided a solely signature-based VES
 - ◆ Defined resolution independence to “separate” this construction from existing ones



Conclusions and open problems

- ◆ Existing VES definitions might not capture desired functionality
 - ◆ Provided a solely signature-based VES
 - ◆ Defined resolution independence to “separate” this construction from existing ones
 - ◆ Demonstrated how stronger resolution duplication could be used to construct public-key encryption



Conclusions and open problems

- ◆ Existing VES definitions might not capture desired functionality
 - ◆ Provided a solely signature-based VES
 - ◆ Defined resolution independence to “separate” this construction from existing ones
 - ◆ Demonstrated how stronger resolution duplication could be used to construct public-key encryption
- ◆ Are VES just misnamed? Or would applications fail if encryption part were missing?



P²OFE: Privacy-Preserving Optimistic Fair Exchange of Digital Signatures

SESSION ID: Protocols - CRYPT-R02

Qiong Huang¹, Duncan S. Wong² and Willy Susilo³

¹ South China Agricultural University, Guangzhou, China

² City University of Hong Kong, HK SAR, China

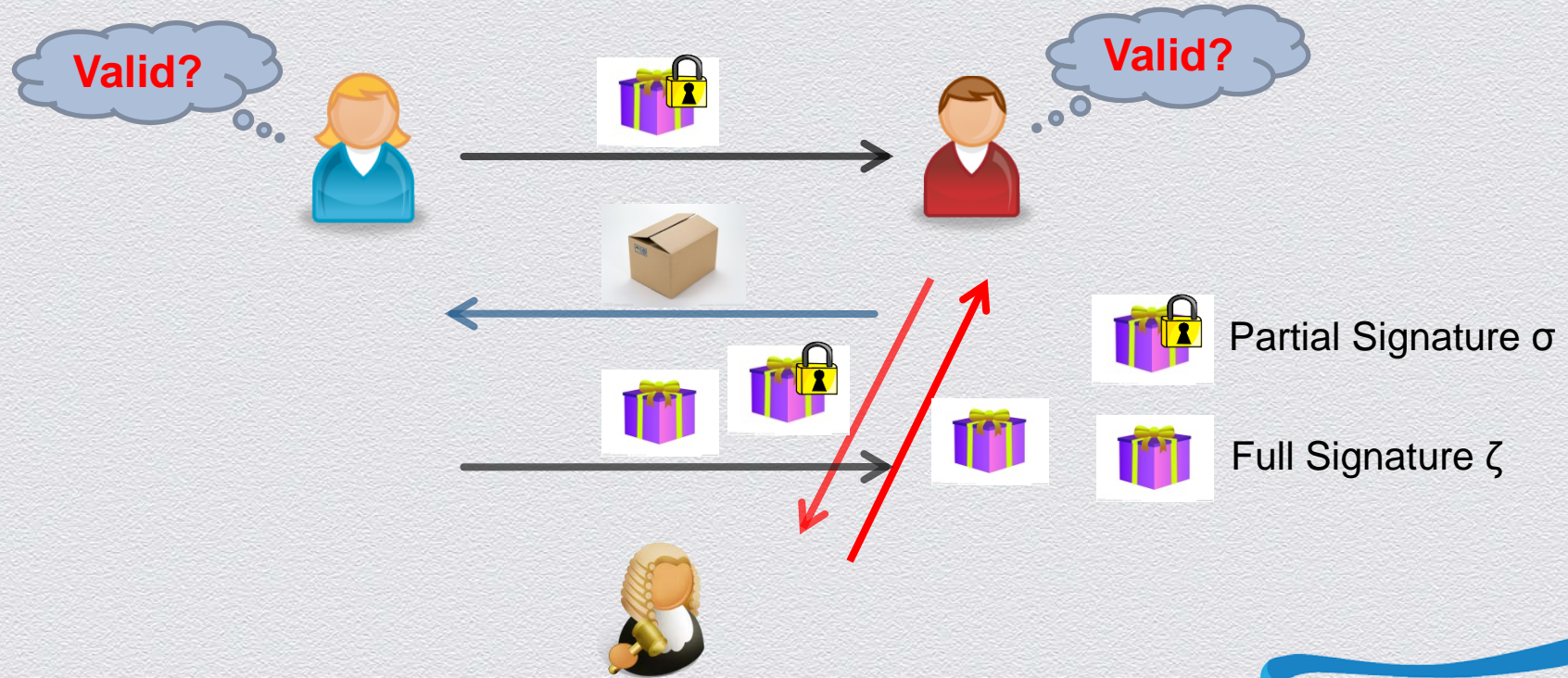
³ University of Wollong, Wollongong, Australia



Fair Exchange

- ◆ Gradual Release of Secret
 - ◆ Bit by bit
 - ◆ Require multiple rounds
- ◆ Optimistic Fair Exchange
 - ◆ Semi-trusted (offline) party
 - ◆ Involved only when there's a dispute

Optimistic Fair Exchange



Optimistic Fair Exchange

- ◆ PKC 2007
 - ◆ Multi-user setting
- ◆ CT-RSA 2008
 - ◆ Chosen-key model
- ◆ Asiacrypt 2008
 - ◆ Ambiguous OFE
- ◆ Pairing 2010, PKC 2012
 - ◆ DCS \rightarrow AOFE
- ◆ Ambiguous OFE
 - ◆ Alice's partial signature reveals her will!
 - ◆ Everyone can verify that σ was generated by Alice.
 - ◆ Bob can show to anybody that Alice is the signer of σ .
- ◆ Solution Idea:
 - ◆ Bob is able to simulate Alice's partial signature

Perfect Ambiguous OFE

- ◆ (A)OFE:
 - ◆ An outsider knows who are involved in an exchange.
- ◆ PAOFE:
 - ◆ No one including the arbitrator can tell from the partial signature who are involved in an exchange.

Y. Wang, M. Au, W. Susilo. *Perfect Ambiguous Optimistic Fair Exchange*. ICICS 2012: 142-153

The Problem We consider

- ◆ In (P)(A)OFE, the arbitrator is able to learn the full signature of Alice.
- ◆ It is not desired in some sensitive applications, and people do not want to put high trust on the arbitrator.

Our Work

- ◆ Introduce the notion of “Privacy-Preserving OFE” (P^2OFE).
- ◆ Present the security models.
- ◆ Propose an efficient construction of P^2OFE .

Even after the resolution, the arbitrator cannot convince others who the signer is.

Our Idea

PK_A, SK_A



PK_B, SK_B



M_A, σ_A

Π

M_B, ζ_B

ζ_A

$\zeta_A \leftarrow \text{Res}^V(SK_B, \theta_A)$

Π : σ_A is either from Alice or from Bob.

M_A, σ_A
 θ_A



APK, ASK

$\theta_A \leftarrow \text{Res}^A(ASK, \sigma_A)$

Definition of P²OFE

- ◆ **PMGen**: system parameter generation \rightarrow (PM)
- ◆ **Setup^{TTP}**: arbitrator key generation \rightarrow (APK, ASK)
- ◆ **Setup^{User}**: user key generation \rightarrow (Pk, Sk)
- ◆ **Psig / Pver**: partial signature (σ) generation / verification
- ◆ **Sig / Ver**: full signature (ζ) generation / verification
- ◆ **Res^A**: resolution by the arbitrator (step 1) $\rightarrow \theta$
- ◆ **Res^V**: resolution by the verifier (step 2) $\rightarrow \zeta$

Definition of P²OFE

- ◆ Resolution Ambiguity Without ASK, anyone cannot tell whether a partial signature was generated by A or simulated by B.
- ◆ Signer Ambiguity
- ◆ Perfect Ambiguity Without SK of the verifier, anyone including the arbitrator cannot tell who is the signer of a given partial signature.
- ◆ Security against Signers
- ◆ Security against Verifiers
- ◆ Security against the arbitrator

Our Construction

- ◆ Full signature ζ is BB short signature.
- ◆ Partial signature σ is a 'twisted' double encryption of ζ .
- ◆ Building blocks used:
 - ◆ Boneh-Boyen (fully secure) Signature
 - ◆ Kiltz' Tag-based Public Key Encryption
 - ◆ Strong One-Time Signature

Signature Generation

- ◆ Full signature: $\zeta \leftarrow (g^{1/(x_i + M + y_i \cdot r)}, r)$
- ◆ Partial signature: $\sigma \leftarrow (\underline{c}, \underline{e}, r, \text{otvk}, \delta)$, where

$$\underline{c} = (c_1, c_2, c_4, c_5) \text{ and } \underline{e} = (e_1, e_2, e_3, e_4, e_5)$$

$$\begin{aligned} c_1 &= F_j^{s'}, \quad c_2 = G_j^{t'}, \quad S = g^{1/(x_i + M + y_i \cdot r)}, \\ e_1 &= F^s, \quad e_2 = G^t, \quad e_3 = Sg^{s+t}g^{s'+t'}, \quad \alpha = H(c_1, c_2, e_1, e_2, e_3, \text{otvk}), \\ c_4 &= (g^\alpha K_j)^{s'}, \quad c_5 = (g^\alpha L_j)^{t'}, \quad e_4 = (g^\alpha K)^s, \quad e_5 = (g^\alpha L)^t, \\ \delta &= \text{OTS.Sig}(\text{otsk}, M \parallel \text{Pk}_i \parallel \text{Pk}_j \parallel \underline{c} \parallel \underline{e} \parallel r), \end{aligned}$$

Signature Verification

- ◆ Full signature: $e(\zeta, X_i g^M Y_i^r) = e(g, g)$
- ◆ Partial signature:

$$\hat{e}(e_4, F) = \hat{e}(e_1, g^\alpha K),$$

$$\hat{e}(e_5, G) = \hat{e}(e_2, g^\alpha L),$$

$$\hat{e}(c_4, F_j) = \hat{e}(c_1, g^\alpha K_j),$$

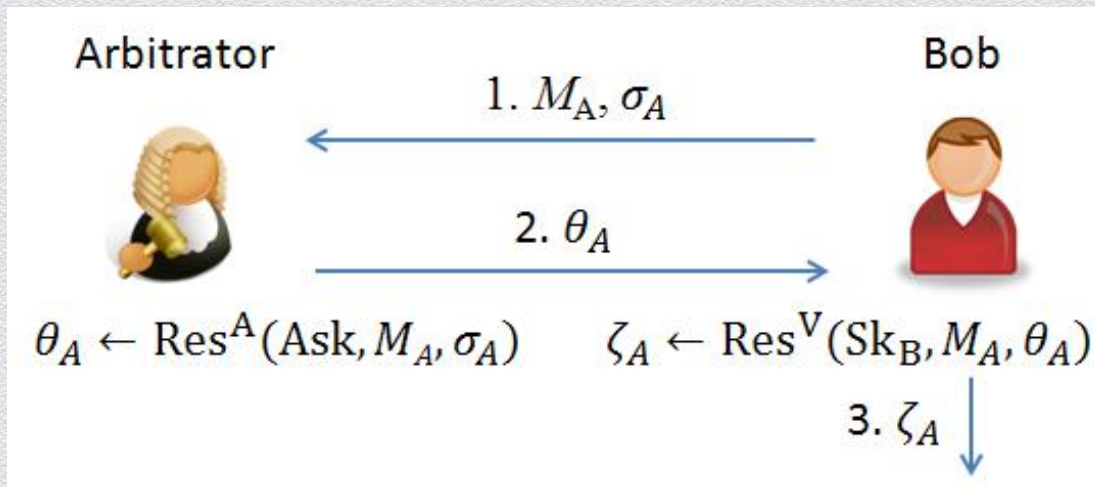
$$\hat{e}(c_5, G_j) = \hat{e}(c_2, g^\alpha L_j),$$

$$\text{OTS.Sig}(M \parallel \text{Pk}_i \parallel \text{Pk}_j \parallel c \parallel e \parallel r, \text{otvk}, \delta) = 1,$$

$$\begin{aligned} \Pi \stackrel{\text{def}}{=} PK \Big\{ (s, t, s', t') : & c_1 = F_j^{s'} \wedge c_2 = G_j^{t'} \wedge e_1 = F^s \wedge e_2 = G^t \\ & \wedge (\hat{e}(e_3 \cdot g^{-s-t-s'-t'}, X_i g^M Y_i^r) = \hat{e}(g, g) \\ & \vee \hat{e}(e_3 \cdot g^{-s-t-s'-t'}, X_j g^M Y_j^r) = \hat{e}(g, g)) \Big\}. \end{aligned}$$

Resolution

- ◆ Arbitrator: $c_3 \leftarrow e_3 e_1^{-\xi_1} e_2^{-\xi_2}$, return $\theta := (c_1, c_2, c_3, c_4, c_5, r, \text{otvk})$
- ◆ Verifier: $S \leftarrow c_3 c_1^{-\xi_{j1}} c_2^{-\xi_{j2}}$, return $\zeta := (S, r)$



Security

◆ Our P²OFE protocol is secure if

1. DLIN assumption holds;
2. SDH assumption holds;
3. H is collision resistant;
4. OTS is one-time strongly unforgeable; and
5. π is sound and witness indistinguishable.

Signer Ambiguity: 1, 3, 4

Perfect Ambiguity: 1, 3, 4

Security against Signers: 2, 5

Security against Arbitrator: 2

RSACONFERENCE2014

FEBRUARY 24 – 28 | MOSCONE CENTER | SAN FRANCISCO



Q&A

Thanks!

2-Pass Key Exchange Protocols From CPA-Secure KEM

Kaoru Kurosawa

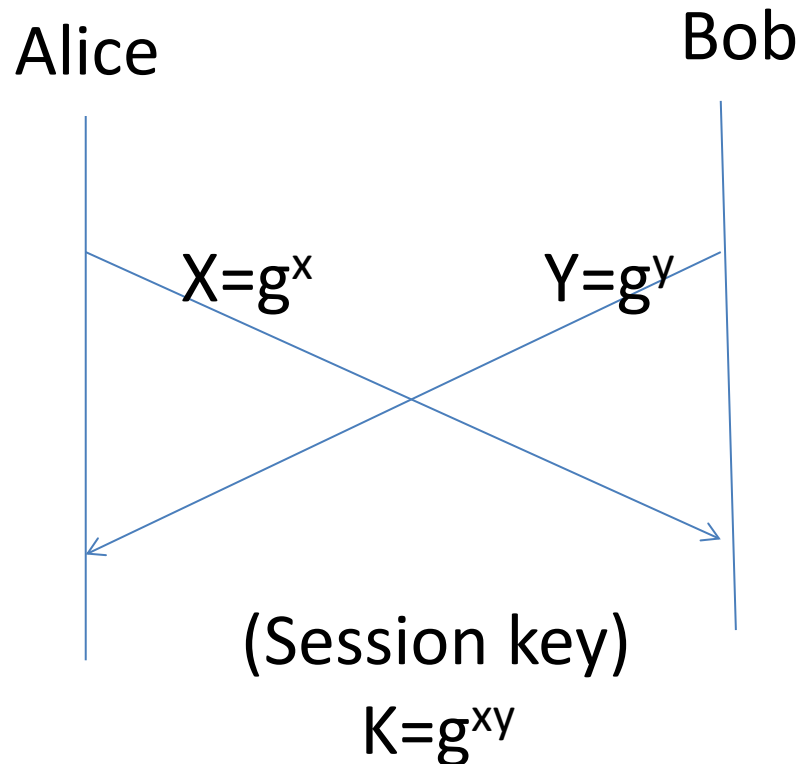
Ibaraki University, Japan

Jun Furukawa

NEC Corporation, Japan

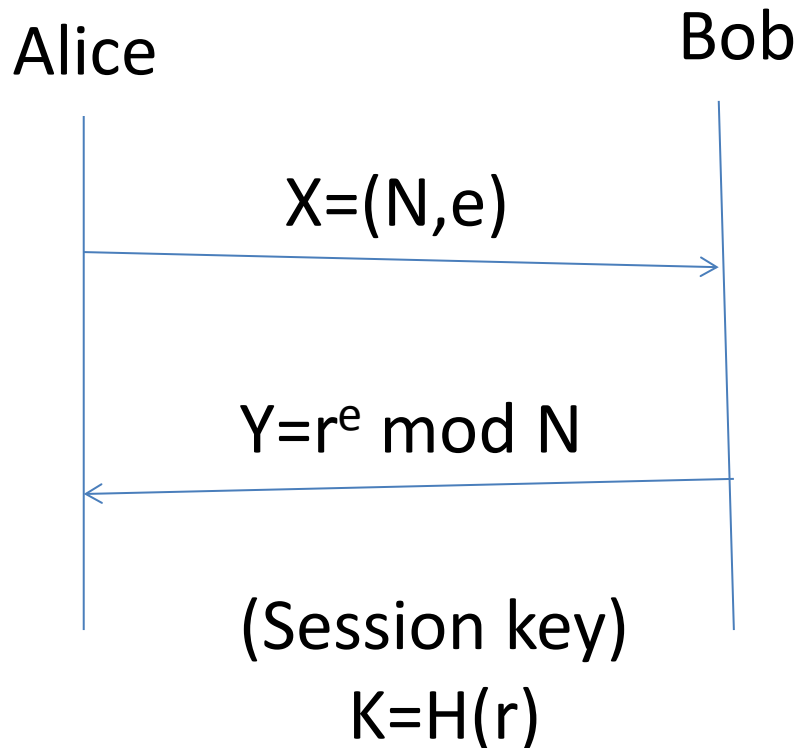
In a **1 round** KE protocol,

Each party sends one message
simultaneously.



In a **2-pass** KE protocol,

Each party sends one message
sequentially.



Most of

- The provably secure KE protocols are based on the **DDH** assumption or the **CDH** assumption

On the other hand,

	round	wPFS	Assumption
Boyd et al.	1-round protocol	×	by using CCA-KEM

A CCA-secure KEM is more generic
than specific number theoretic assumptions.

KEM

- Consists of (Gen, Enc, Dec).
- In particular,
Enc(pk) outputs a ciphertext c and the key K which is used for a symmetric-key encryption scheme.

A KEM is CPA-secure if

- No adversary can distinguish between (c, K) and (c, random)

A KEM is CCA-secure if

- No adversary can distinguish between
(c, K) and (c, random)
even if
the adversary can query $c' \neq c$
to the decryption oracle

For example,

- Let

$$pk=g^x \text{ and } sk=x$$

$$c=g^r$$

$$K=(pk)^r$$

- This KEM is **CPA**-secure
under the DDH assumption

Cramer-Shoup KEM

- is CCA-secure
under the DDH assumption

Boyd et al. also showed

	round	wPFS	By using
Boyd et al.	1-round	×	CCA-KEM
//	//	○	CCA-KEM + DDH

This construction is **not generic**
because it relies on the DDH assumption

Fujioka et al. showed

	round	wPFS	By using
Boyd et al.	1-round	×	CCA-KEM
//	//	○	CCA-KEM + DDH
Fujioka et al.	2-pass	○	CCA-KEM

We show

	round	wPFS	By using
Fujioka et al.	2-pass	○	CCA-KEM
This paper	2-pass	○	CPA-KEM

Our assumption is weaker
than Fujioka et al.

In fact

We show 3 generic constructions
by using a **CPA**-secure KEM

Proposed	security
1 st one is	CK-secure
2 nd one is	eCK-secure
3 rd one is	Both CK and eCK-secure

In Canetti-Krawczyk (CK) Model

A long-term key

$lsk_A \rightarrow$

Alice

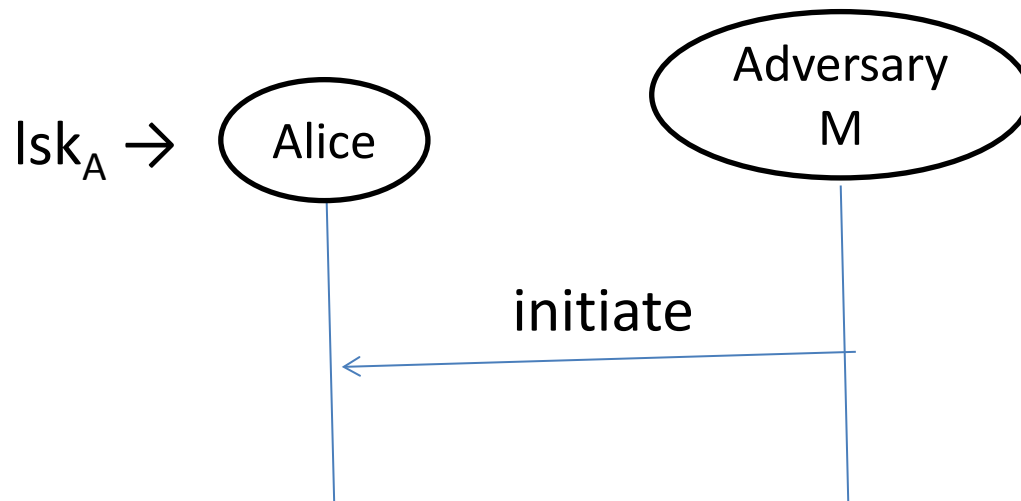


```
graph TD; Alice((Alice)); AdversaryM((Adversary M)); Alice --- AliceLine[ ]; AdversaryM --- AdversaryLine[ ]
```

Adversary

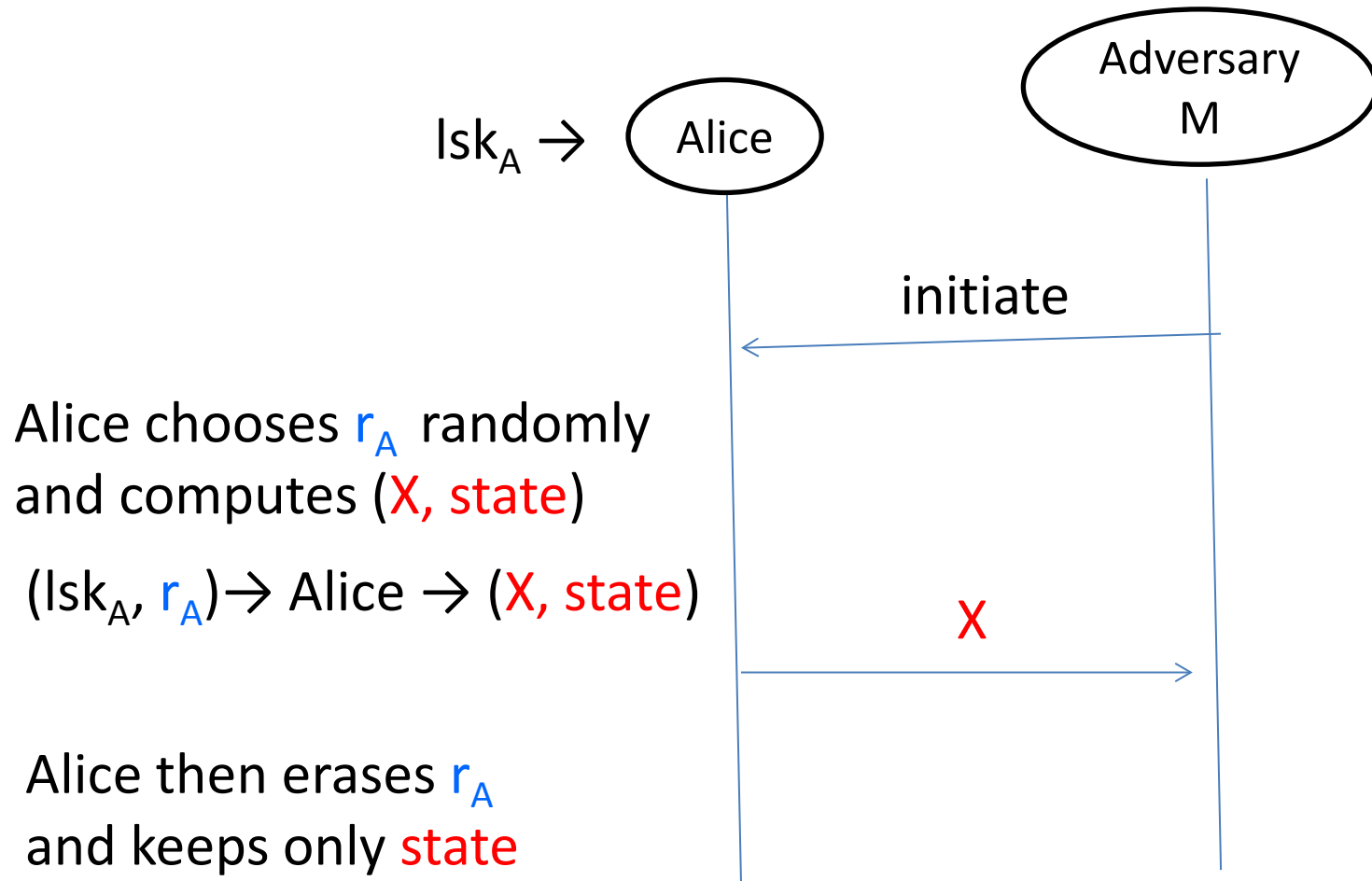
M

First M sends “initiate” to Alice

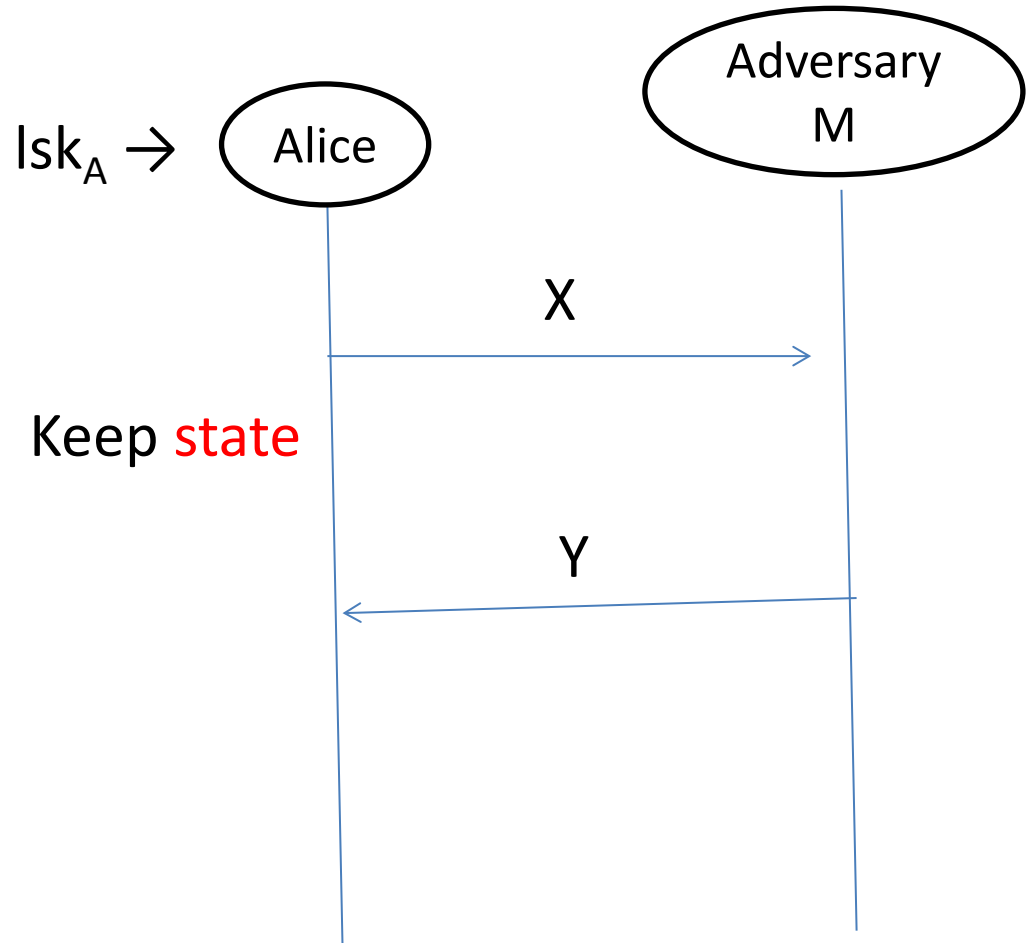


Hi,
I'm Bob.
Let's initiate a session.

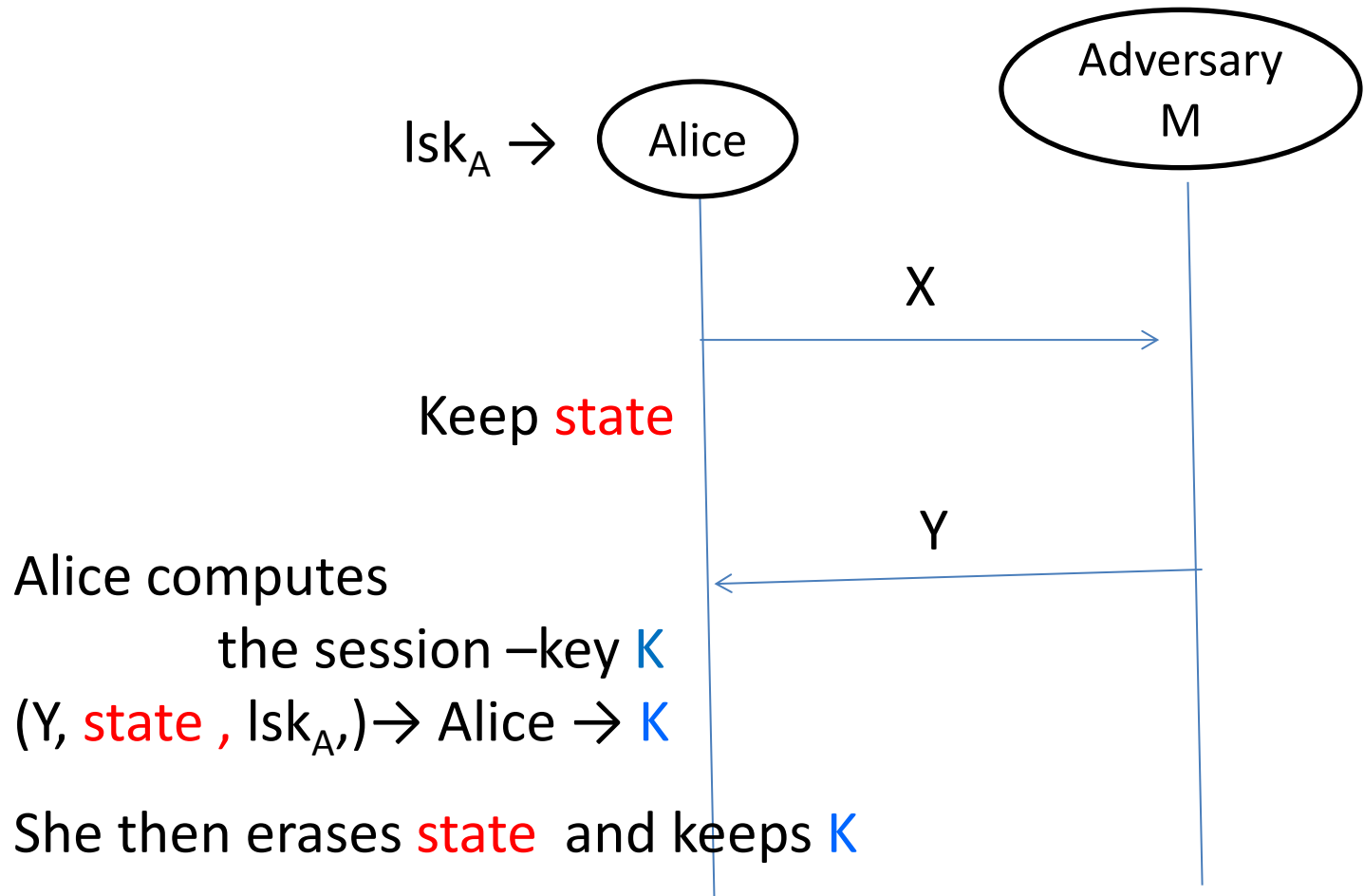
Then



Next M sends “Y” to Alice



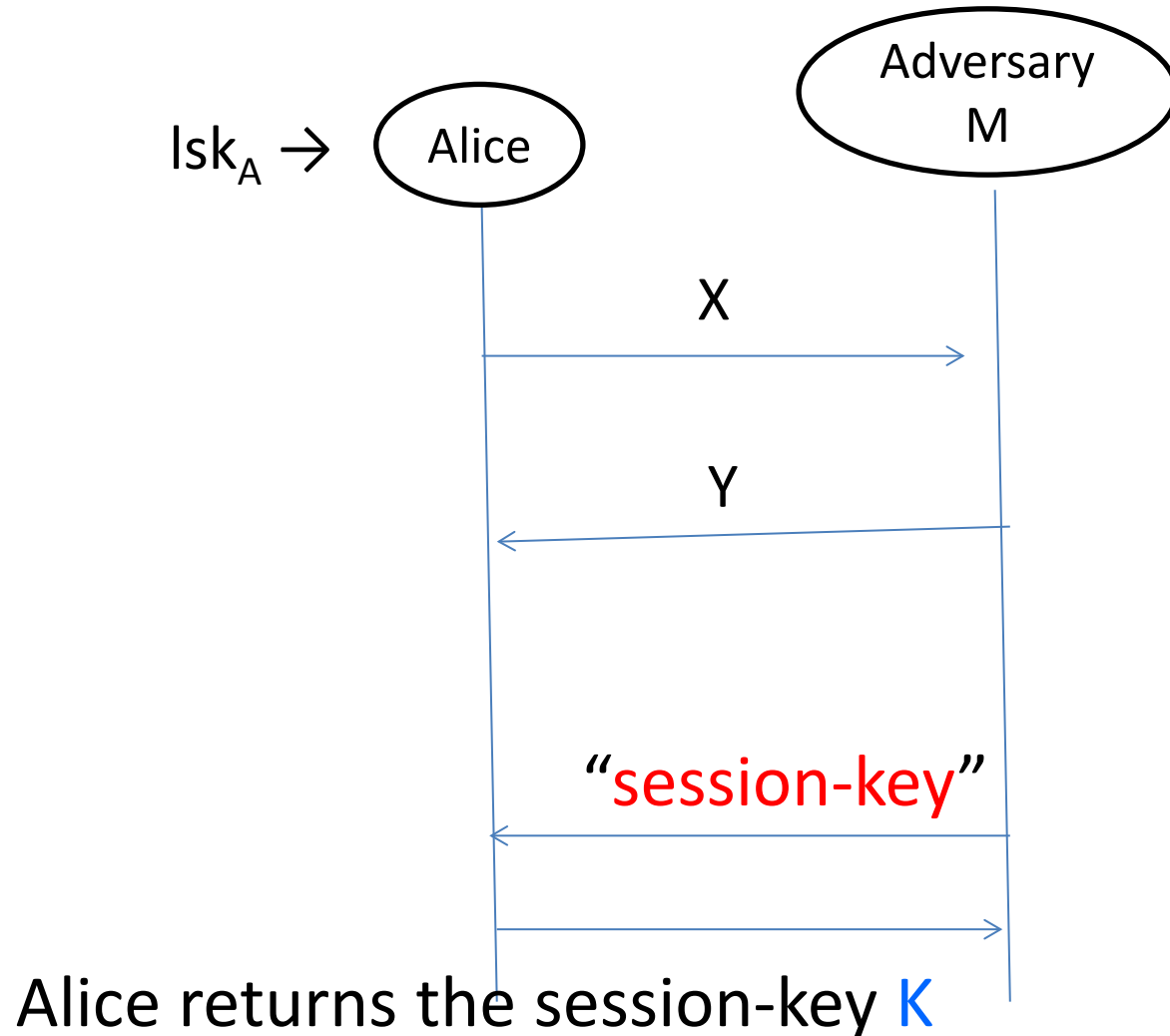
Then



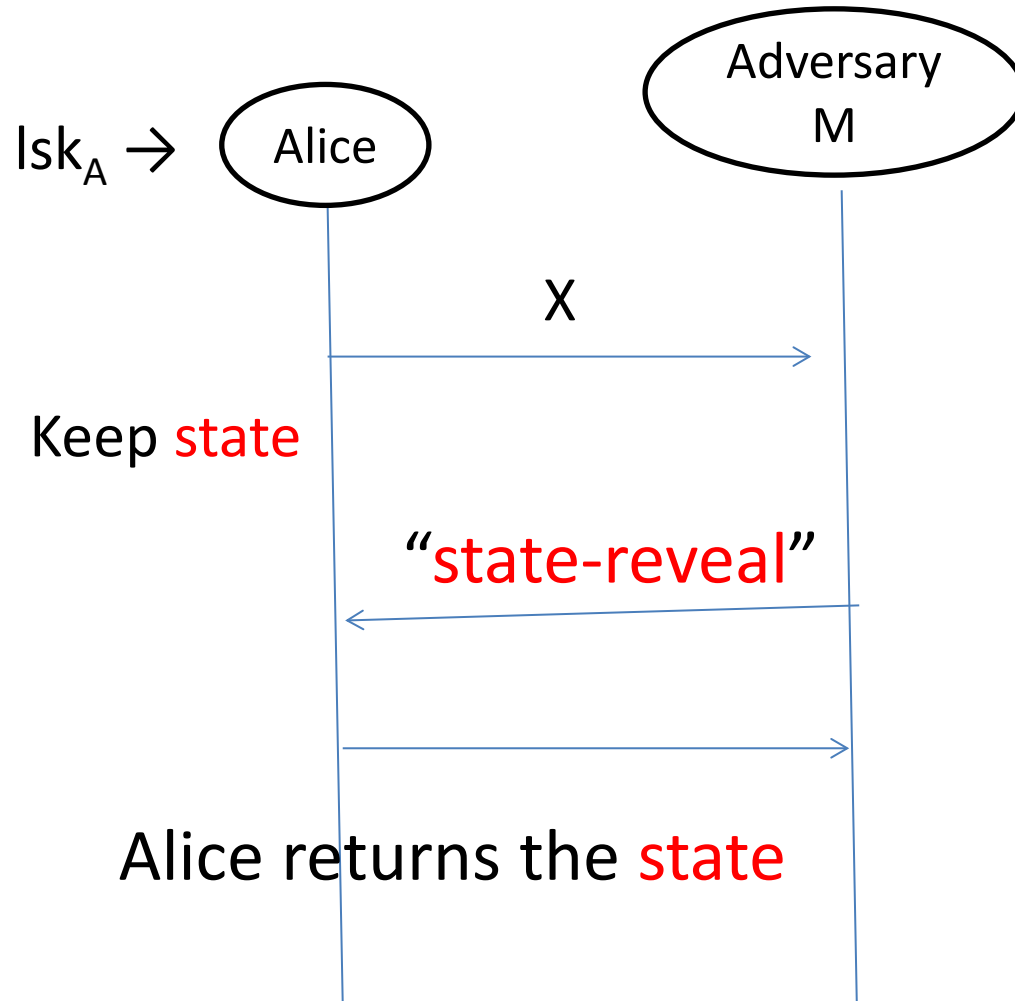
M can issue

- A session-key query
 - A state-reveal query
 - and a corrupt query
- to Alice

For a “session-key” query



For a “state-reveal” query

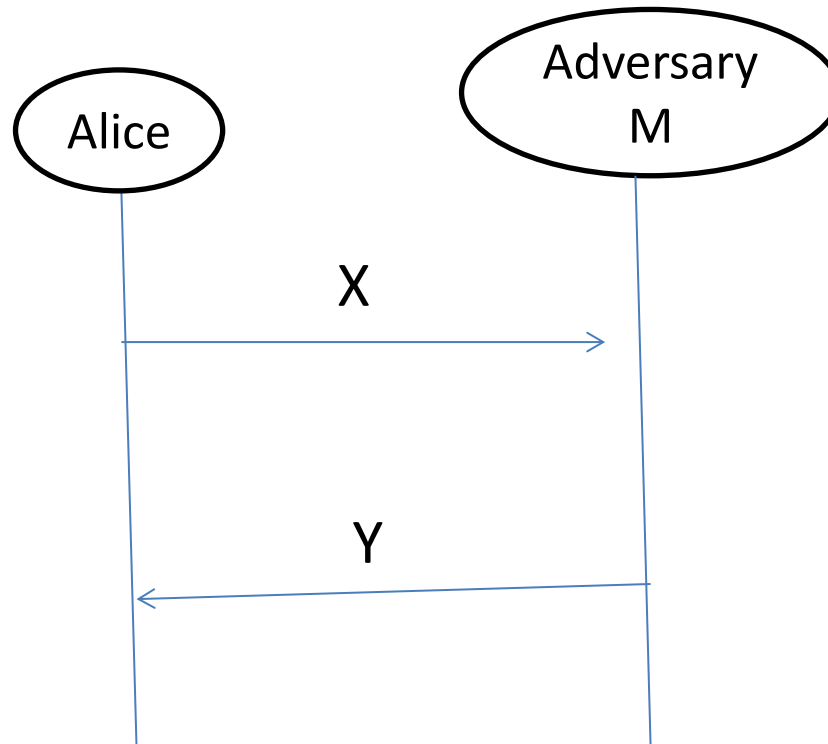


For a “corrupt” query

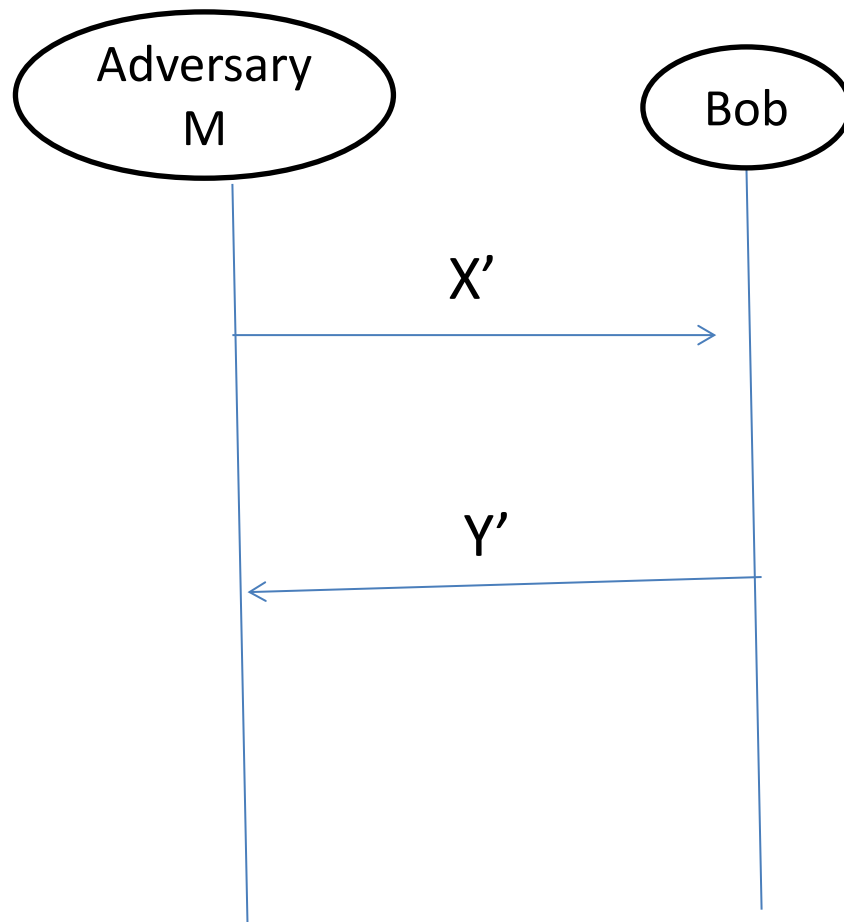
Alice returns

- the long-term key lsk_A ,
- the state
- and all the session keys stored at that time.

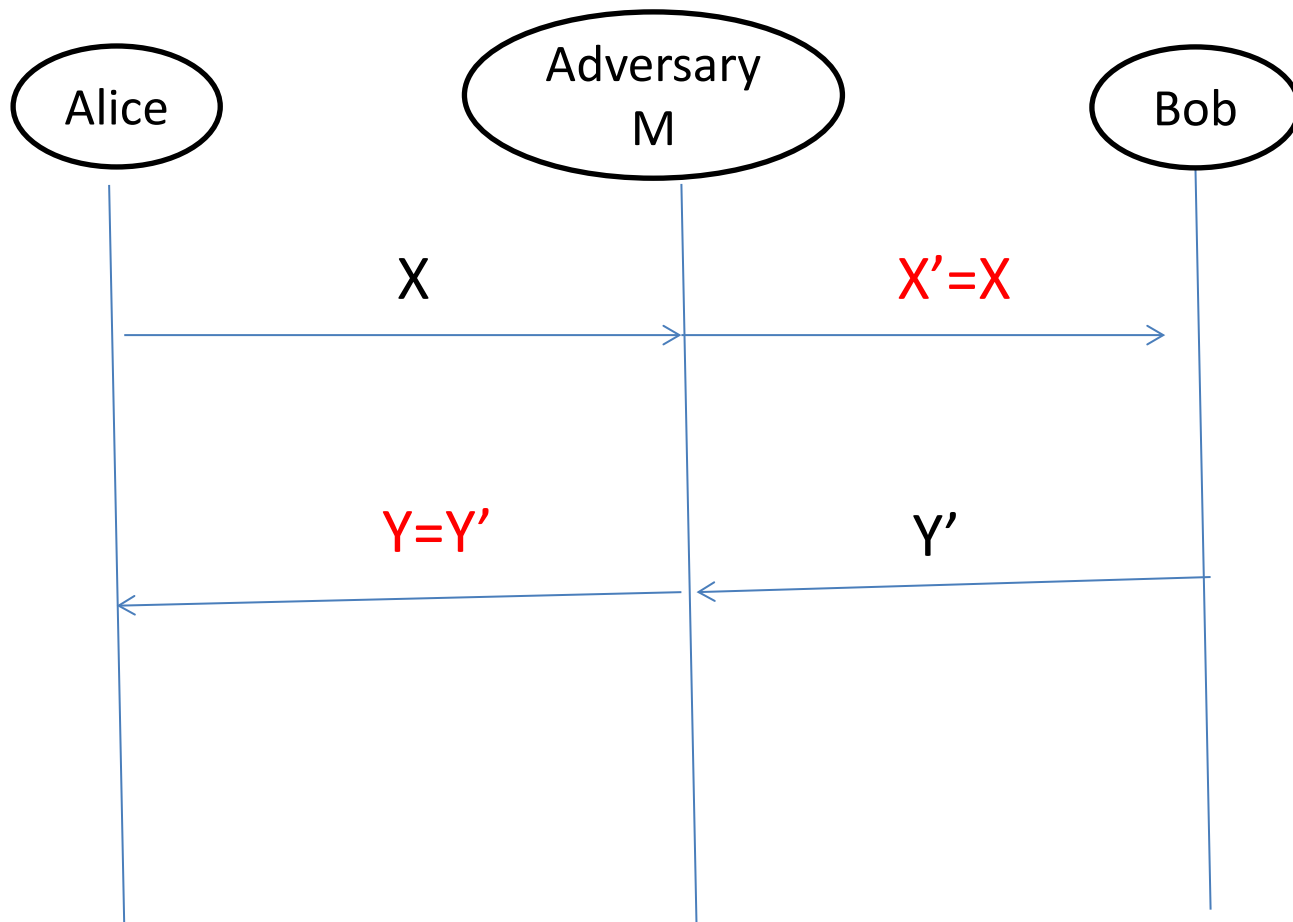
An instance between Alice and M
is called a session



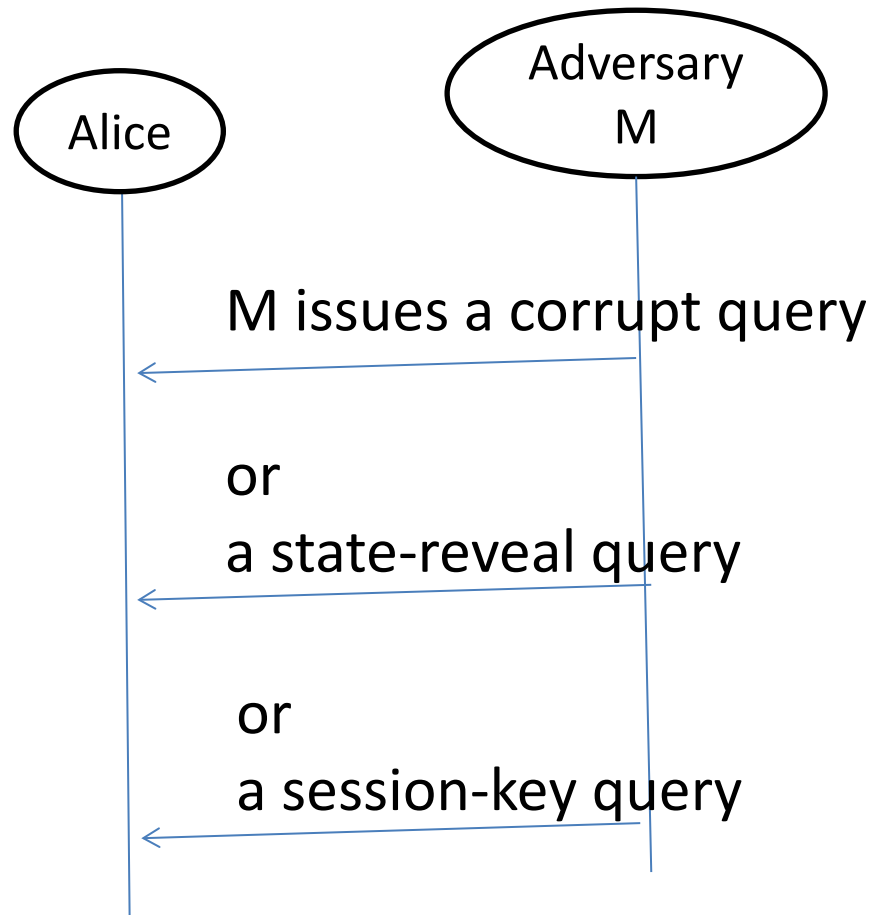
An instance between Bob and M is also a session



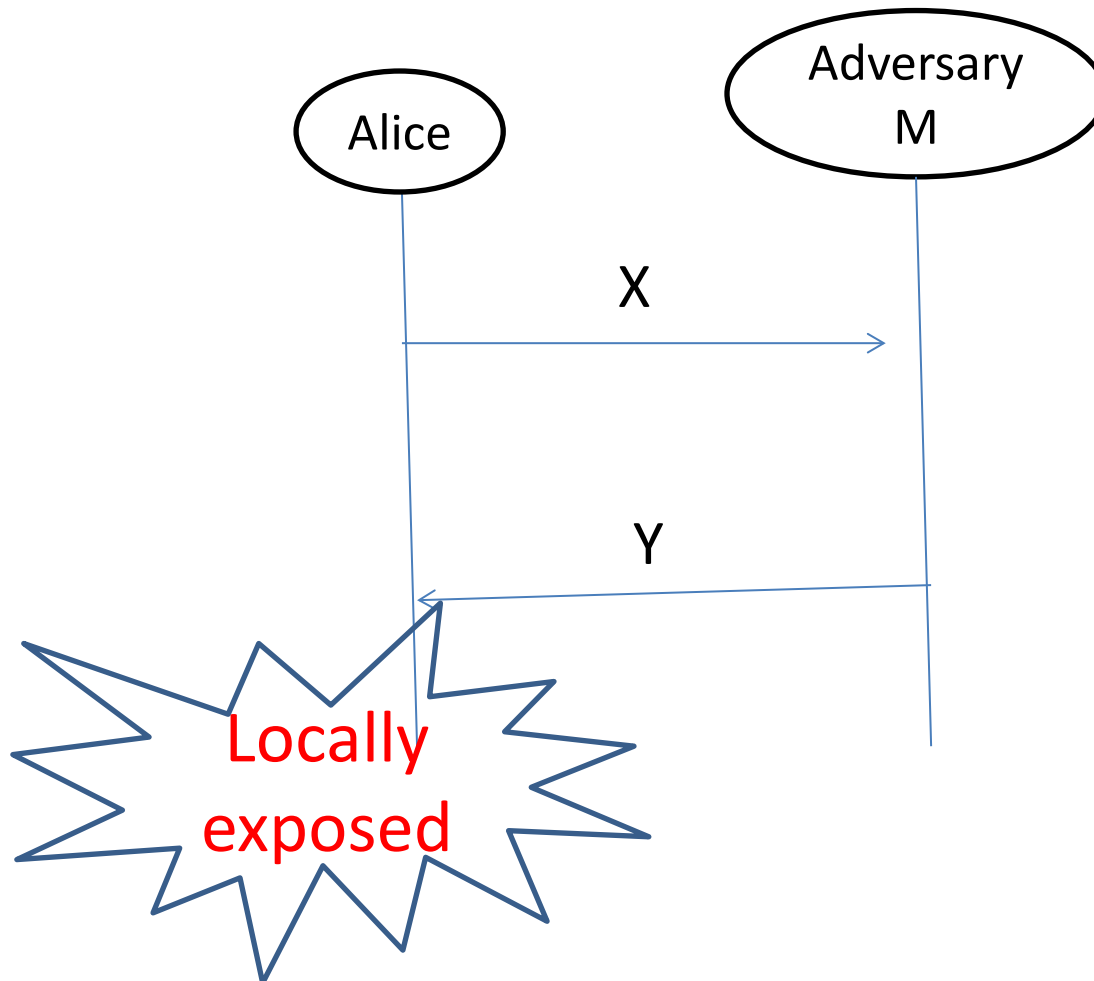
They are matching sessions if



A session is **locally exposed** if

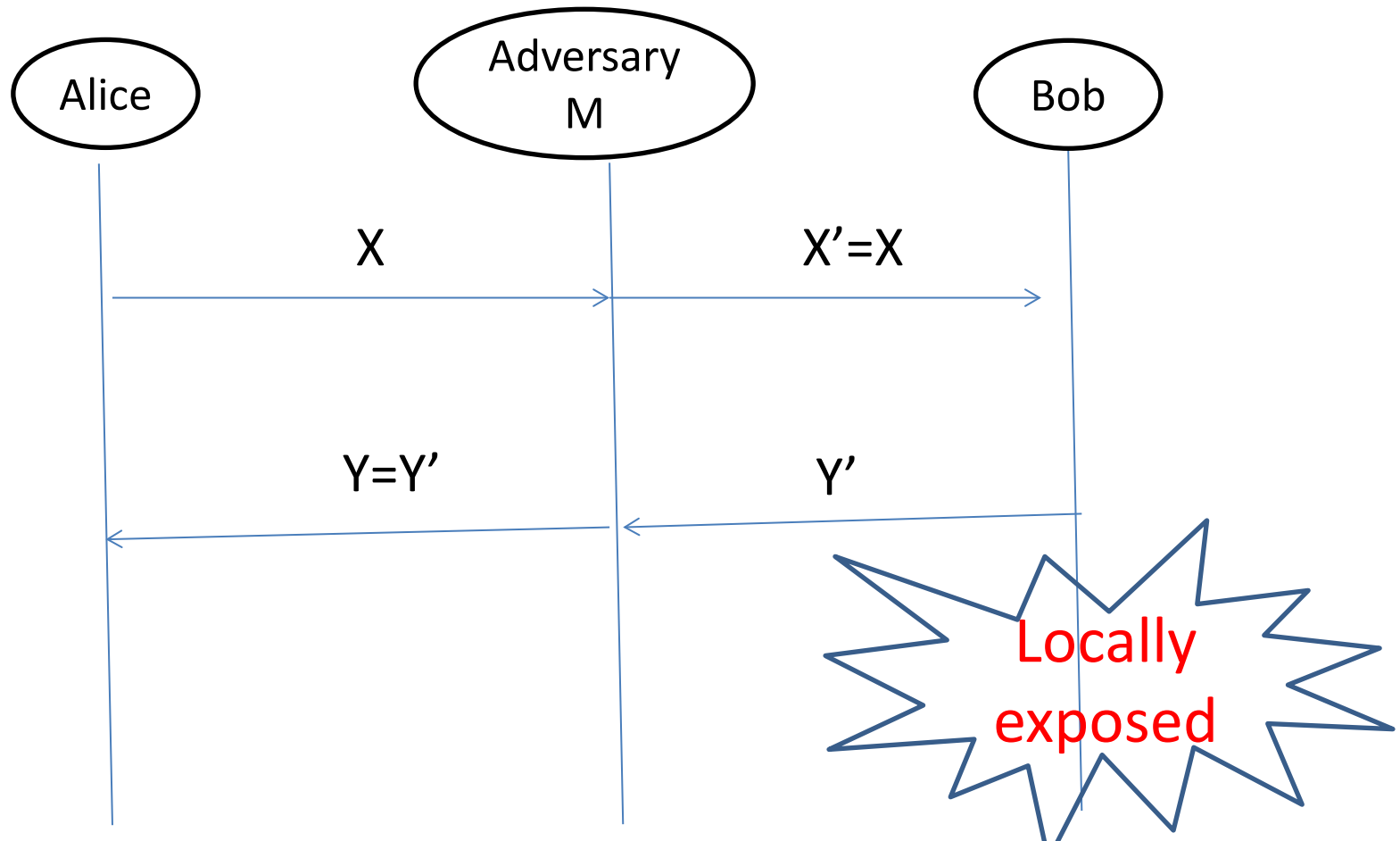


A session is **exposed** (1)
if it is **locally exposed**



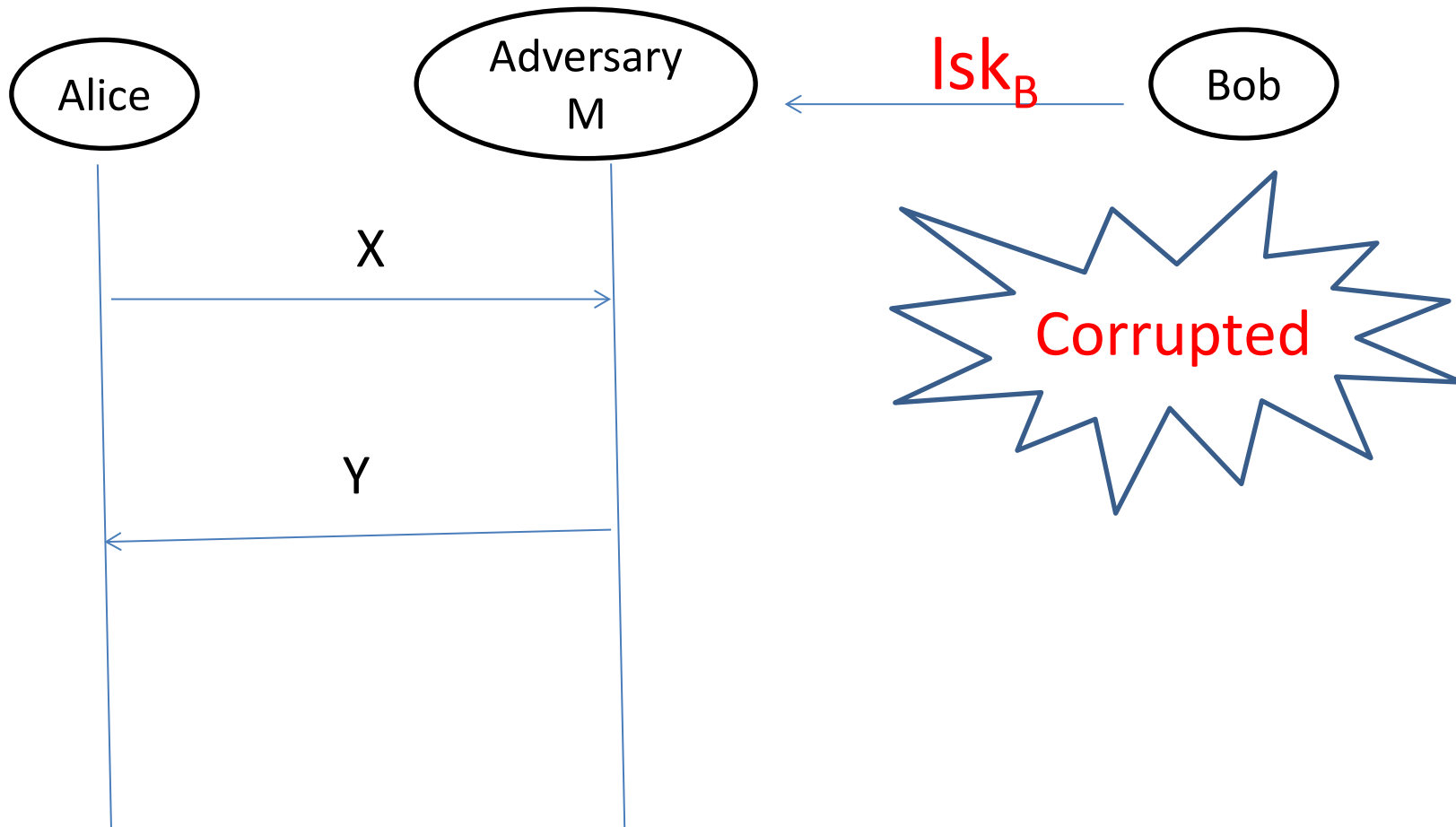
A session is **exposed** (2)

or, it has a matching session
that is **locally exposed**

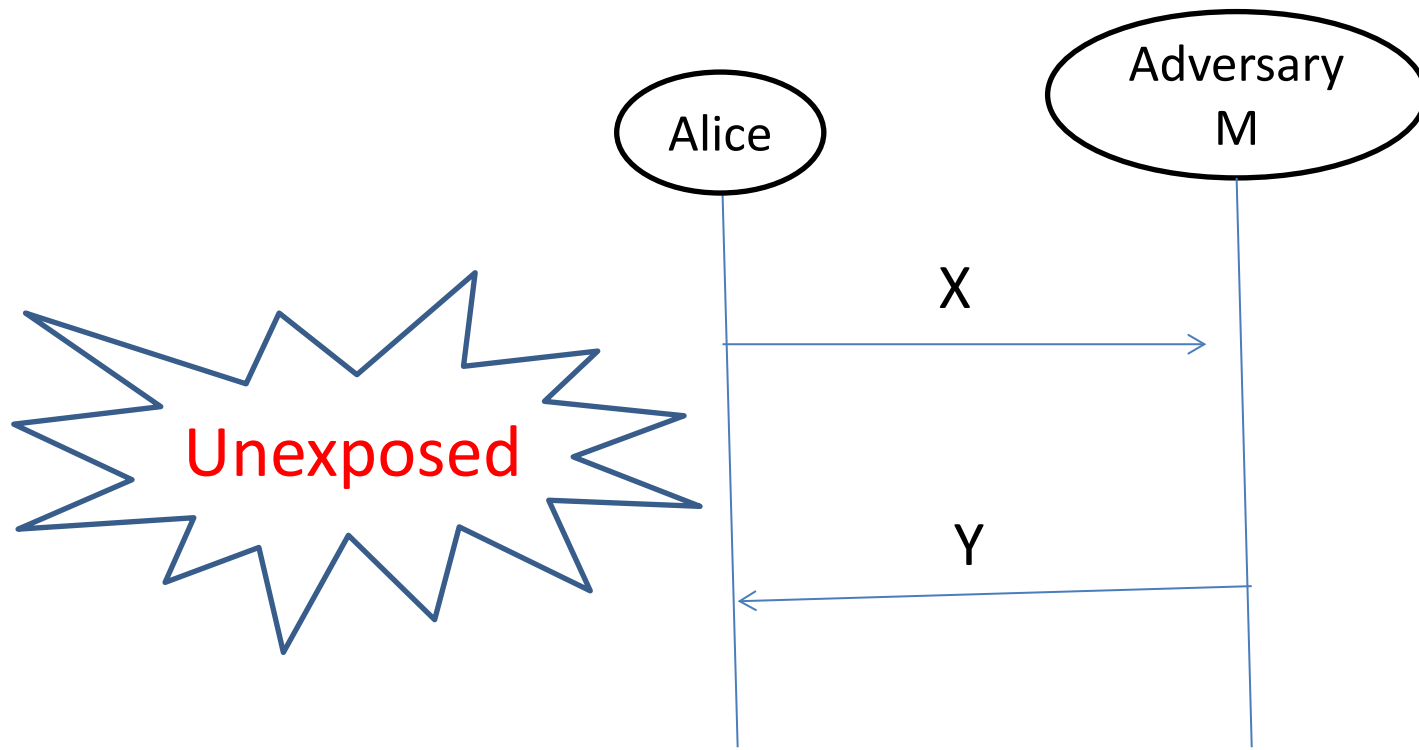


A session is **exposed** (3)

or, it doesn't have a matching session
and Bob is **corrupted**

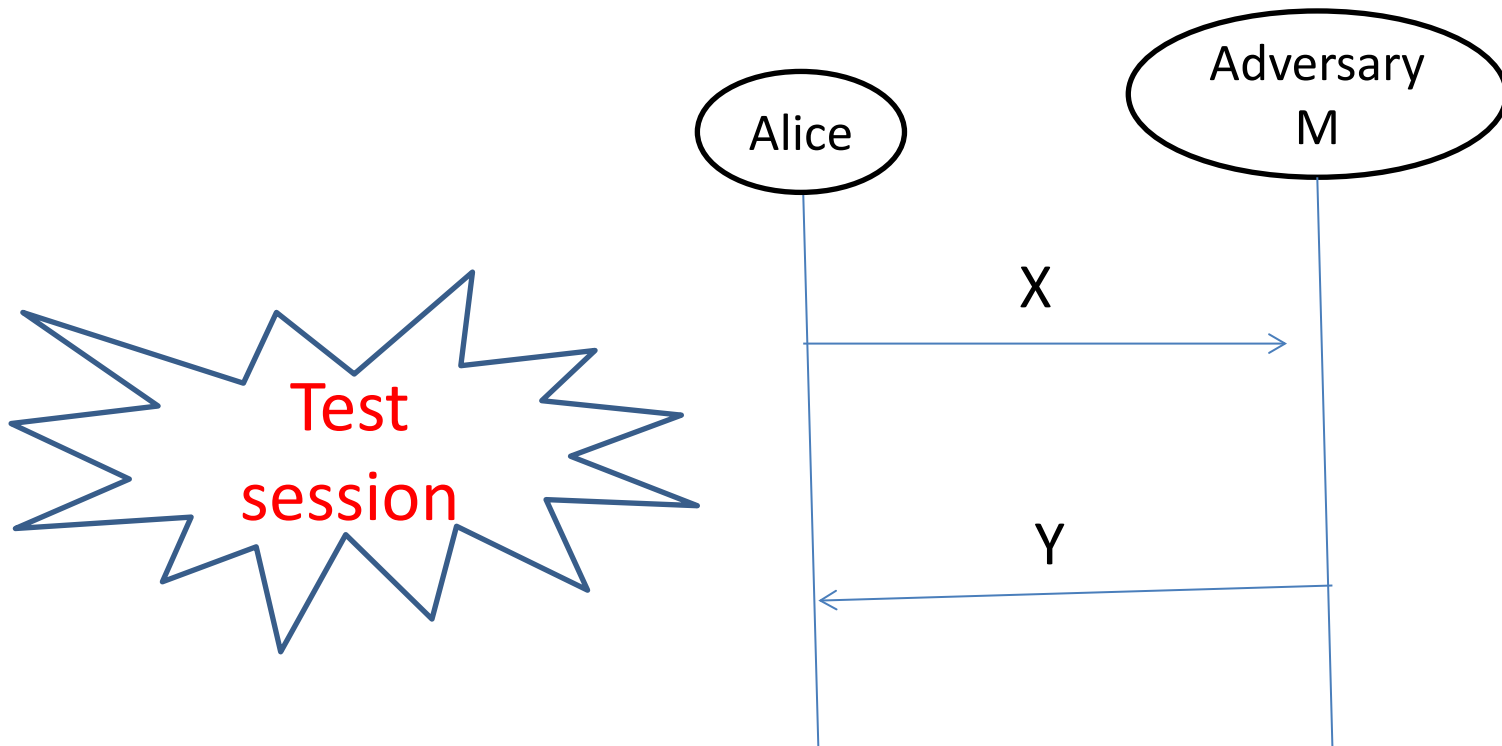


A session which is not exposed
is called **unexposed**



At some point,

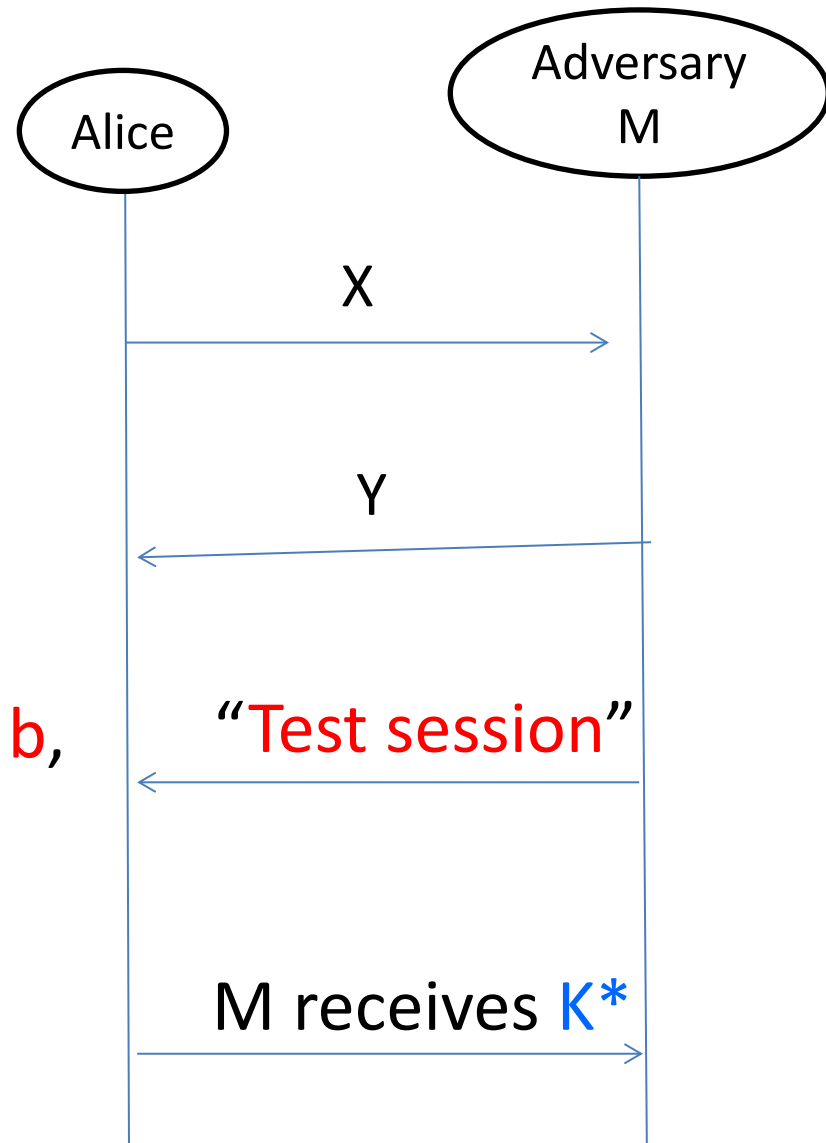
M chooses an **unexposed session**
as a **test session**



Then
we choose a random bit b ,
and let

$K^* = K$ if $b = 0$

$K^* = \text{random}$ if $b = 1$



M finally outputs a bit b'

- The advantage of M is defined as

$$\text{Adv}(M) = 2 \times |\Pr(b'=b) - 1/2|$$

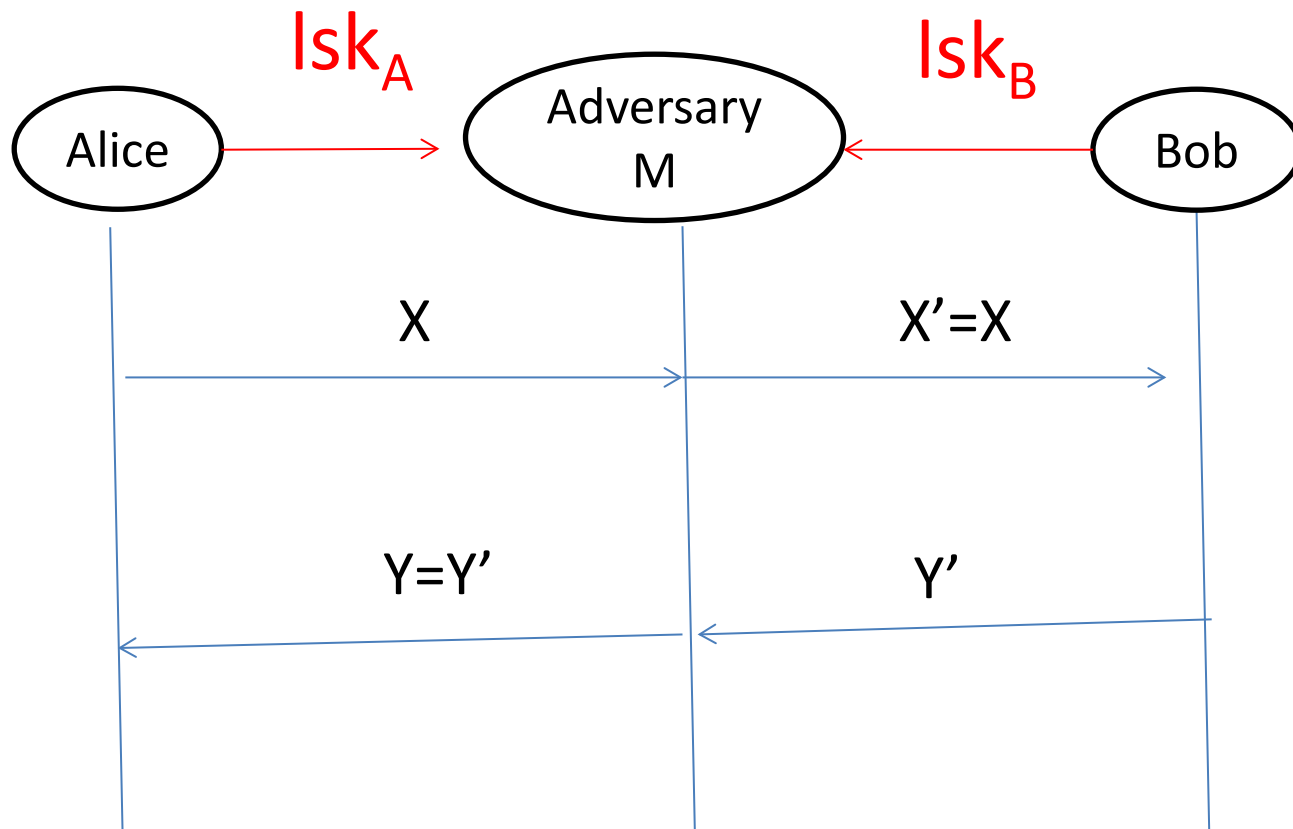
A KE protocol is CK-secure

- If $\text{Adv}(M)$ is negligible
for any PPT adversary M

Perfect Forward Secrecy (PFS)

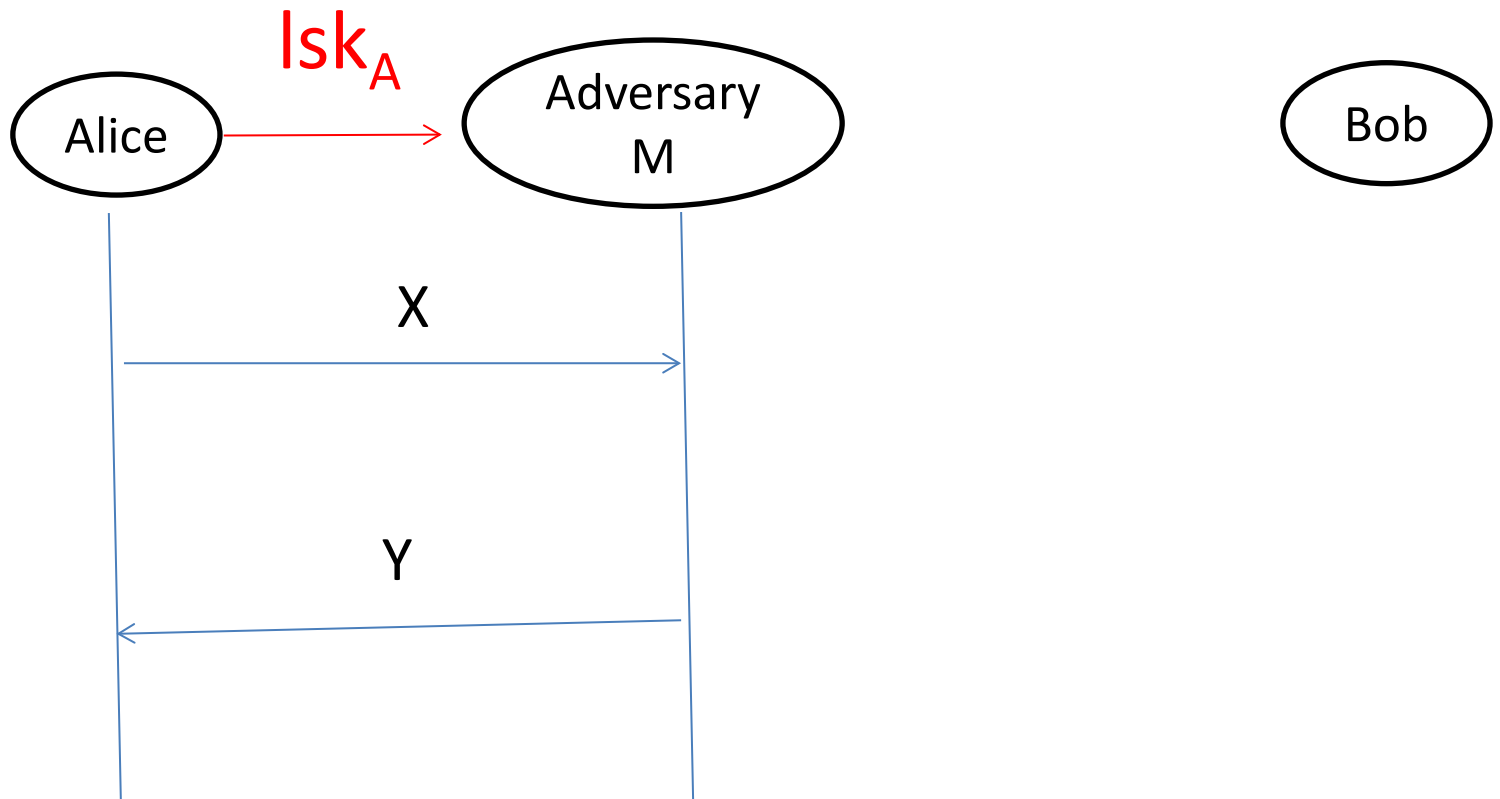
- is defined as follows.
- Suppose that the session key K expired, and it was erased.
- After that,
the adversary can obtain lsk_A and lsk_B .
- PFS requires that
 K should look random even in this case.

If the test-session has a matching session, then M can obtain both lsk_A and lsk_B after the session key K is erased.



In **Weak** PFS (wPFS),

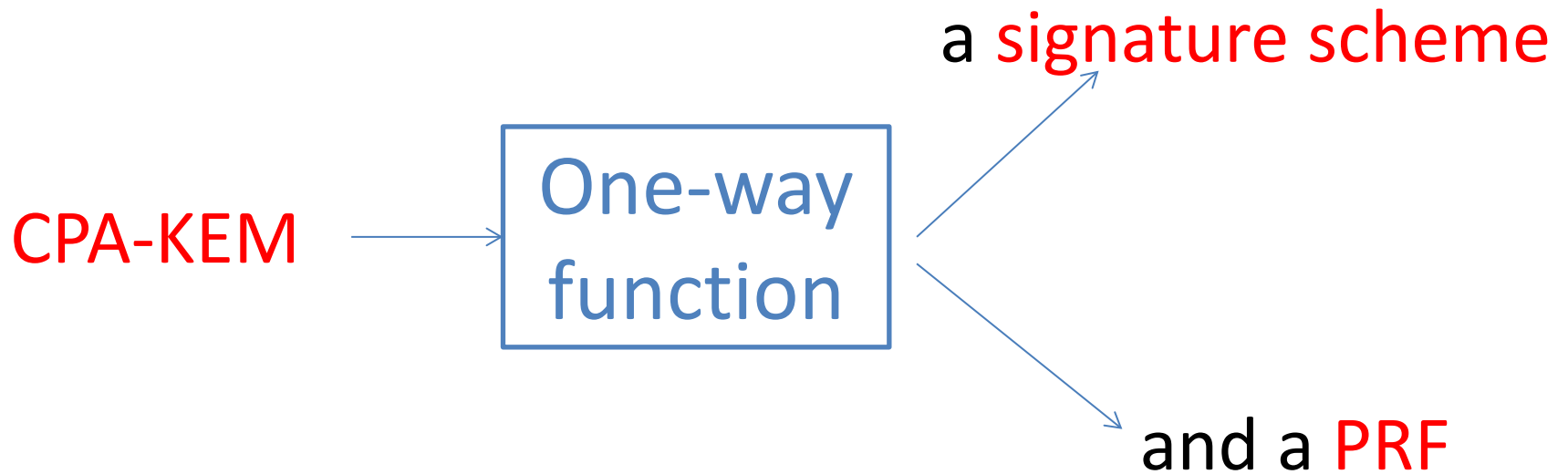
If the test-session doesn't have a matching session, then M cannot obtain sk_B



A KE protocol is CK-secure with wPFS

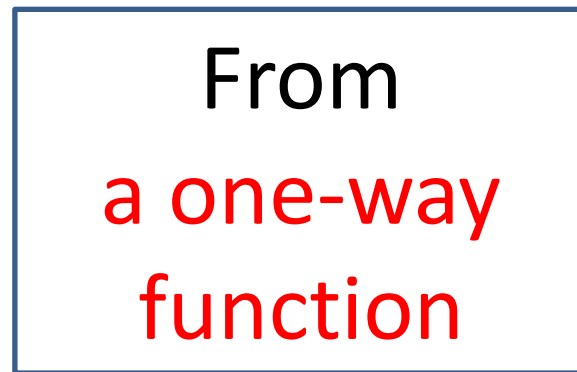
- If $\text{Adv}(M)$ is negligible
for any such PPT adversary M

Our construction uses



We can construct

a signature scheme



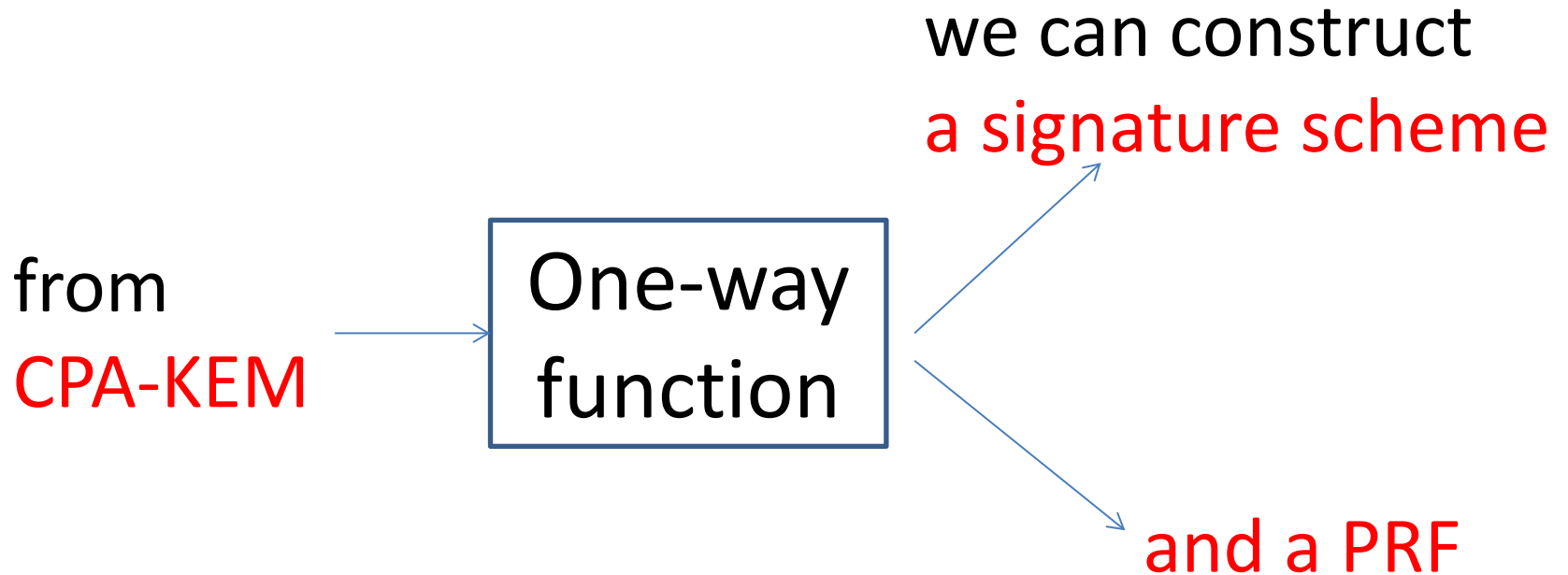
and a PRF

The key generation algorithm Gen of a CPA-secure KEM

- Can be considered as a one-way function from a random string to a public-key pk.

$$\text{Gen}(\text{random}) \rightarrow (\text{pk}, \text{sk})$$

Therefore



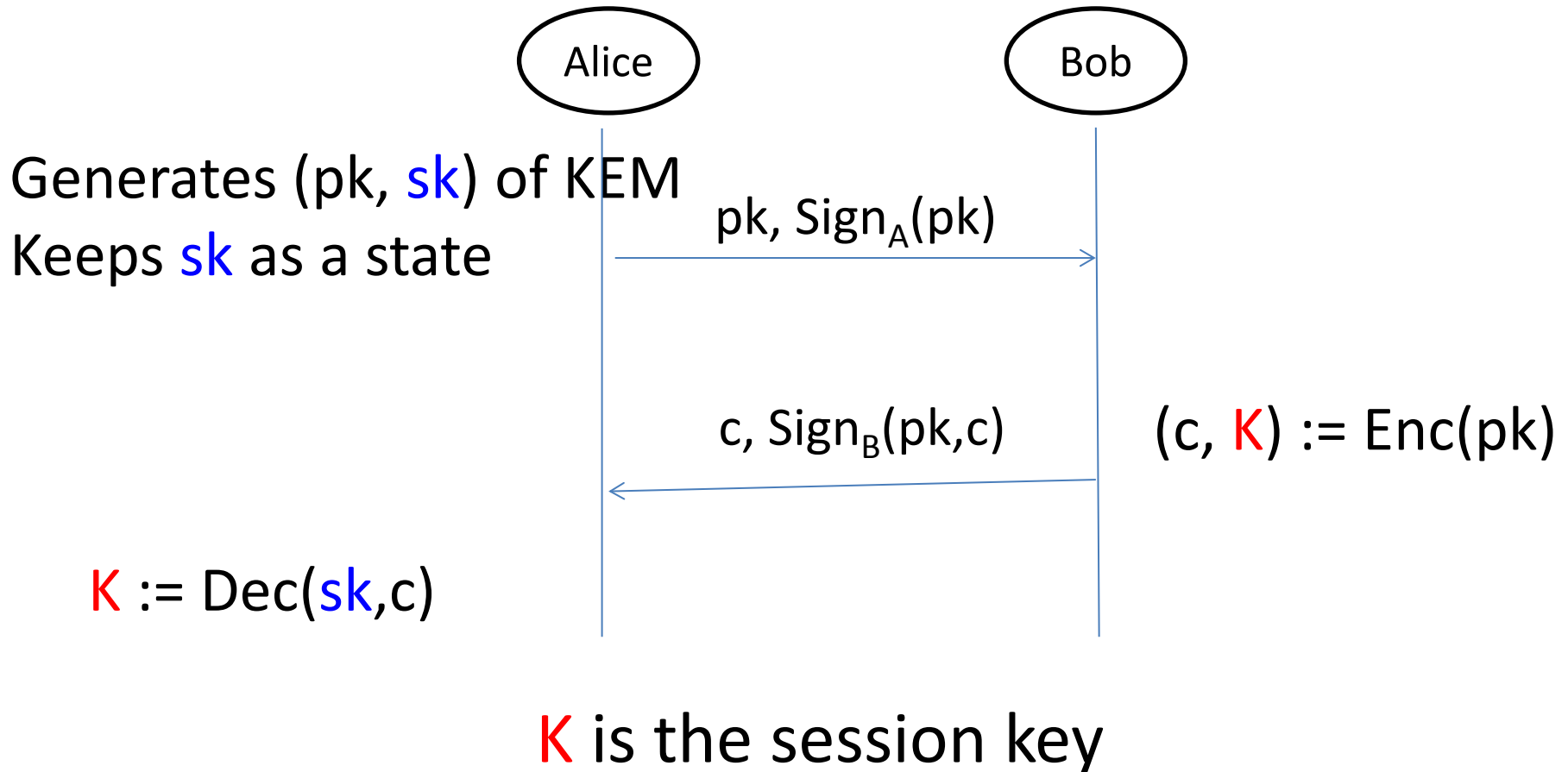
Hence

our minimum assumption is that
there exists a **CPA-secure KEM**

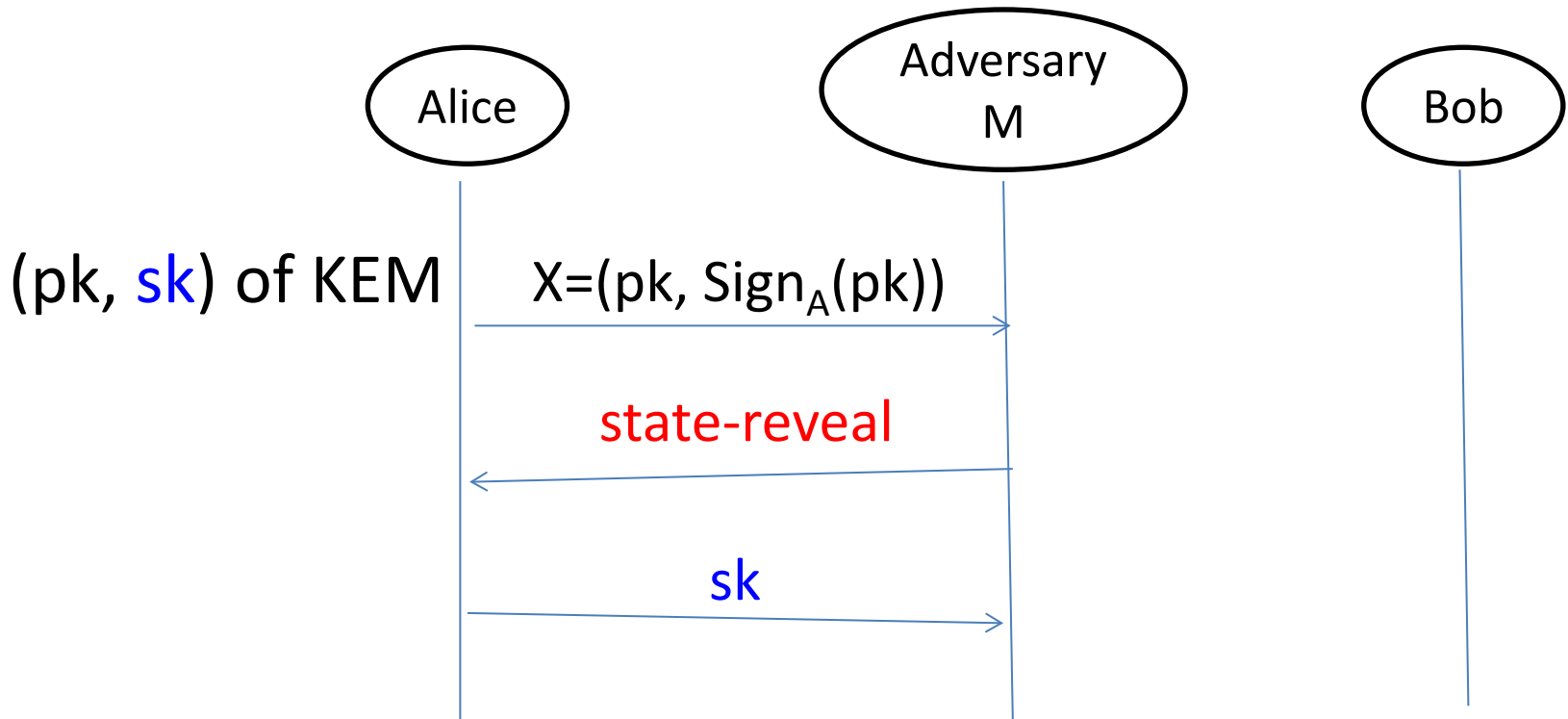
Let

- $\text{KEM}=(\text{Gen}, \text{Enc}, \text{Dec})$
be a CPA-secure KEM
- $\text{SIG}=(G, \text{Sign}, \text{Verify})$
be a signature scheme

In our Naïve approach

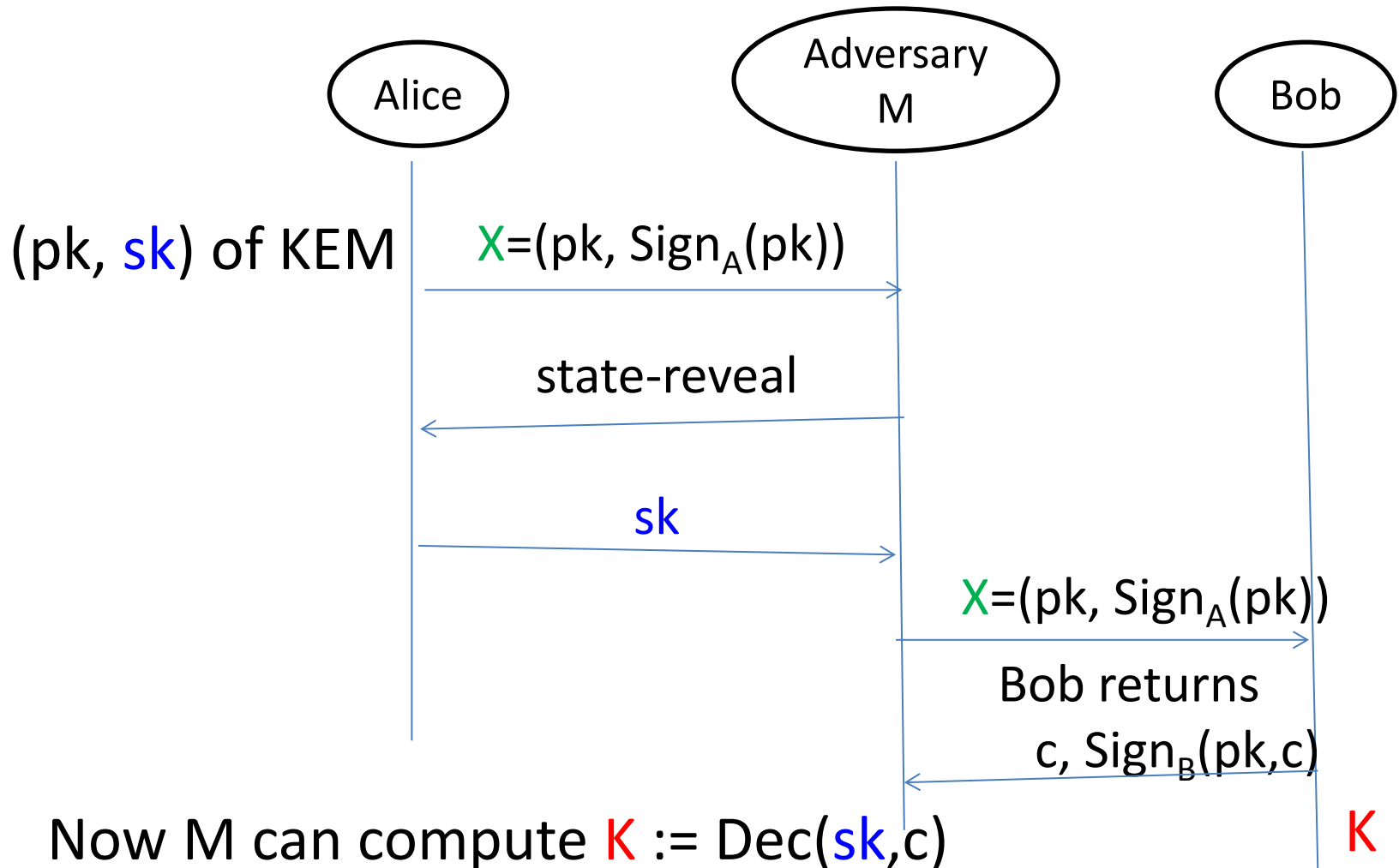


However, there exists an attack



After receiving $X = (pk, \text{Sign}_A(pk))$,
M issues a **state-reveal** query.
Then Alice returns sk

Then M sends **X** to Bob



We overcome this problem

- By using a twisted PRF trick.
- This trick was introduced by Fujioka et al.
- However,
we cannot prove that
their construction has the desired property

So

- We formulate tPRF formally
- and then give a new construction which satisfies our definition.

Our definition of tPRF

We say that $F(k,r)$ is a tPRF if

- If k is a key,
- $F(k,r)$ works as a PRF
- Even if r is used as a key,
 $F(k,r)$ also works as a PRF

Our construction of tPRF

- Let PRF be a pseudorandom function
- Let

$$F((k_1, k_2), (r_1, r_2)) = \text{PRF}_{k_1}(r_1) + \text{PRF}_{r_2}(k_2)$$

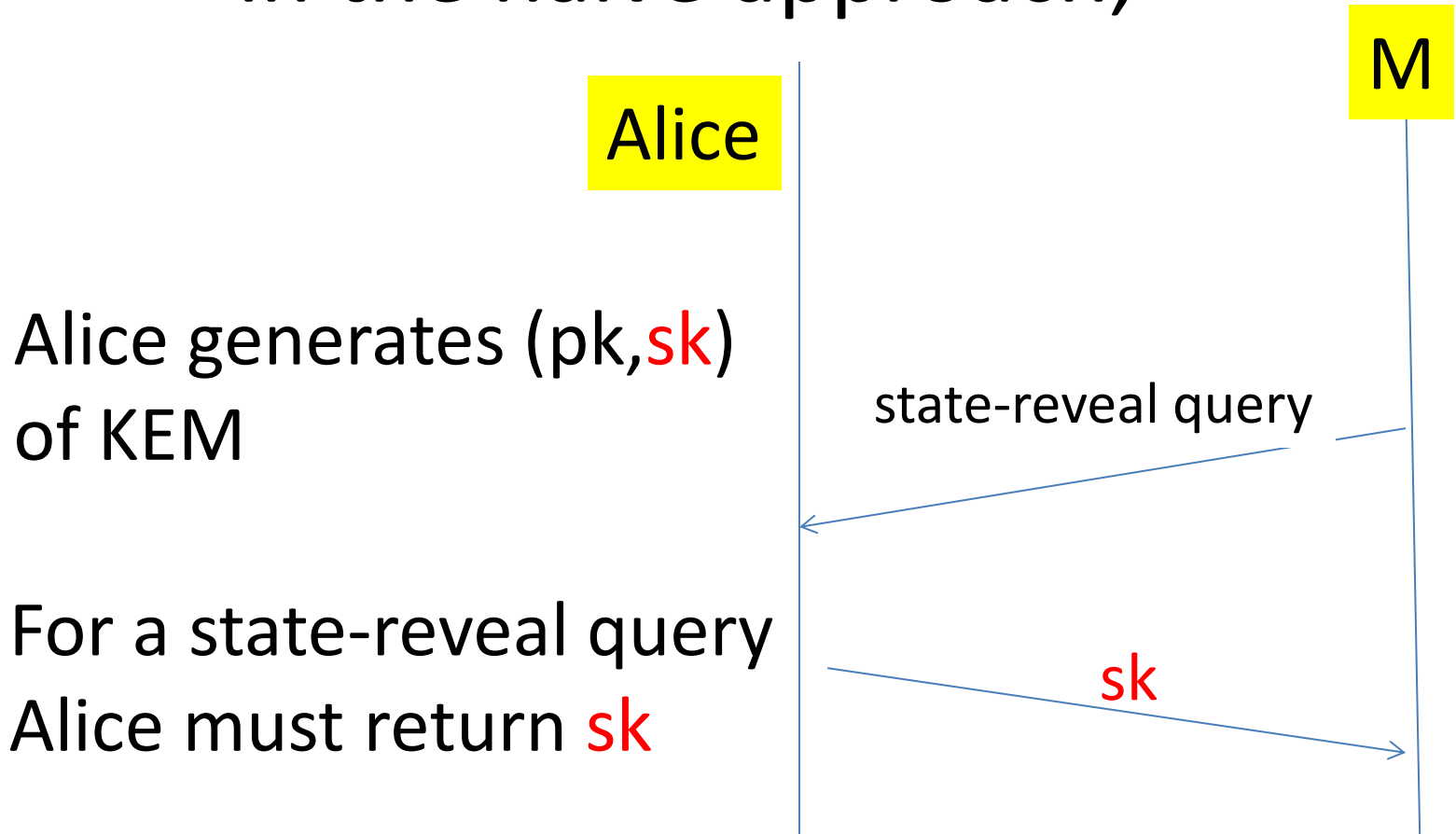
- Then we can prove that
this F is a tPRF.

The construction of Fujioka et al.

$$F(k, (r1, r2)) = \text{PRF}_k(r1) + \text{PRF}_{r2}(k)$$

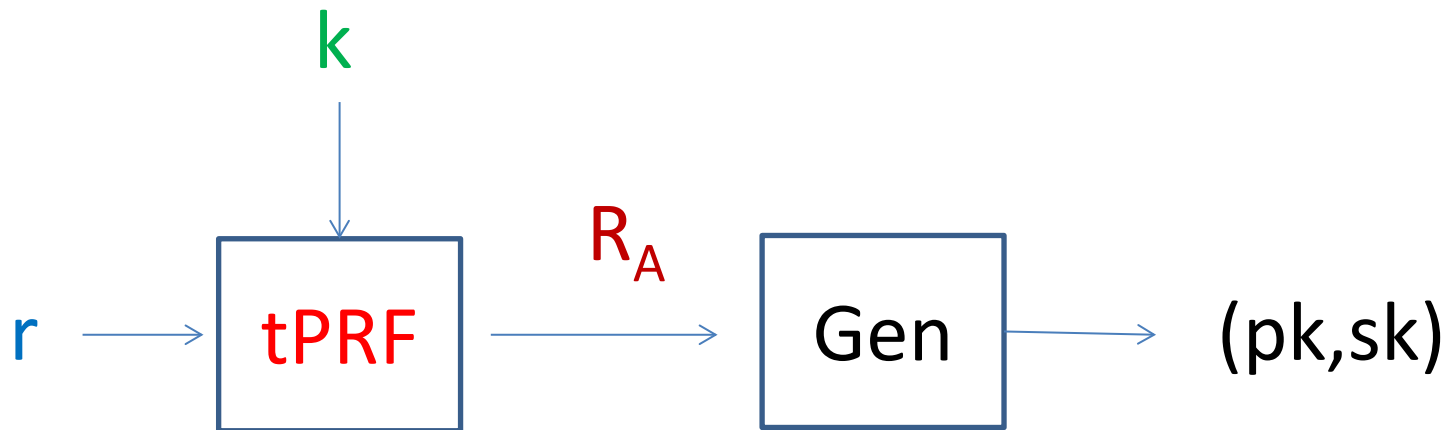
- We **cannot** prove that this F is a tPRF.

Remember that
in the naïve approach,



In the proposed protocol,

Alice has a long-term key

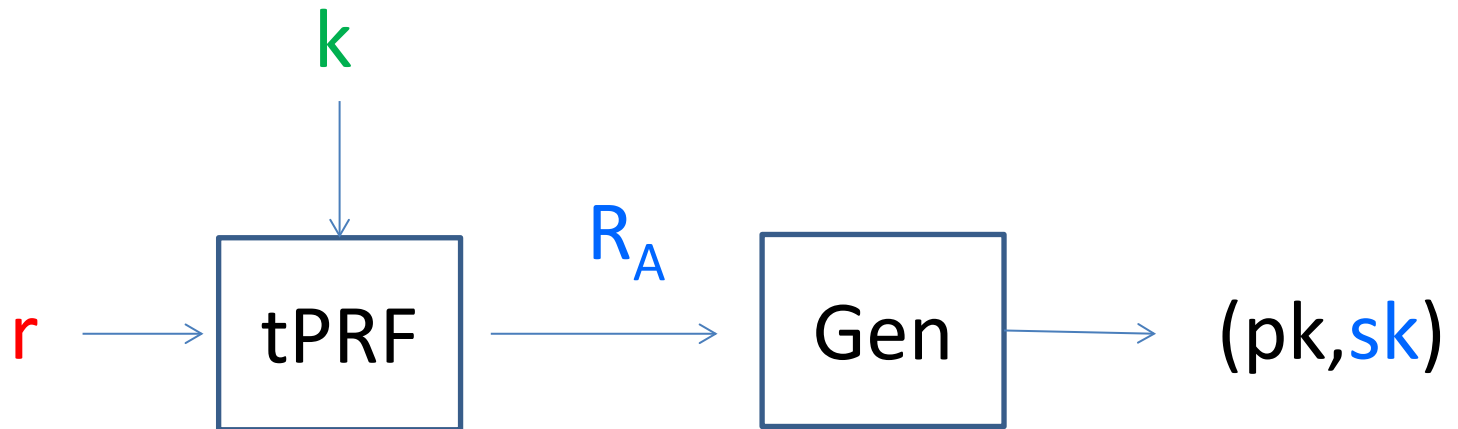


She first chooses **r** randomly
and runs **tPRF** to generate **R_A**

She next runs Gen of KEM
to obtain (pk, sk)

Then Alice erases R_A and sk

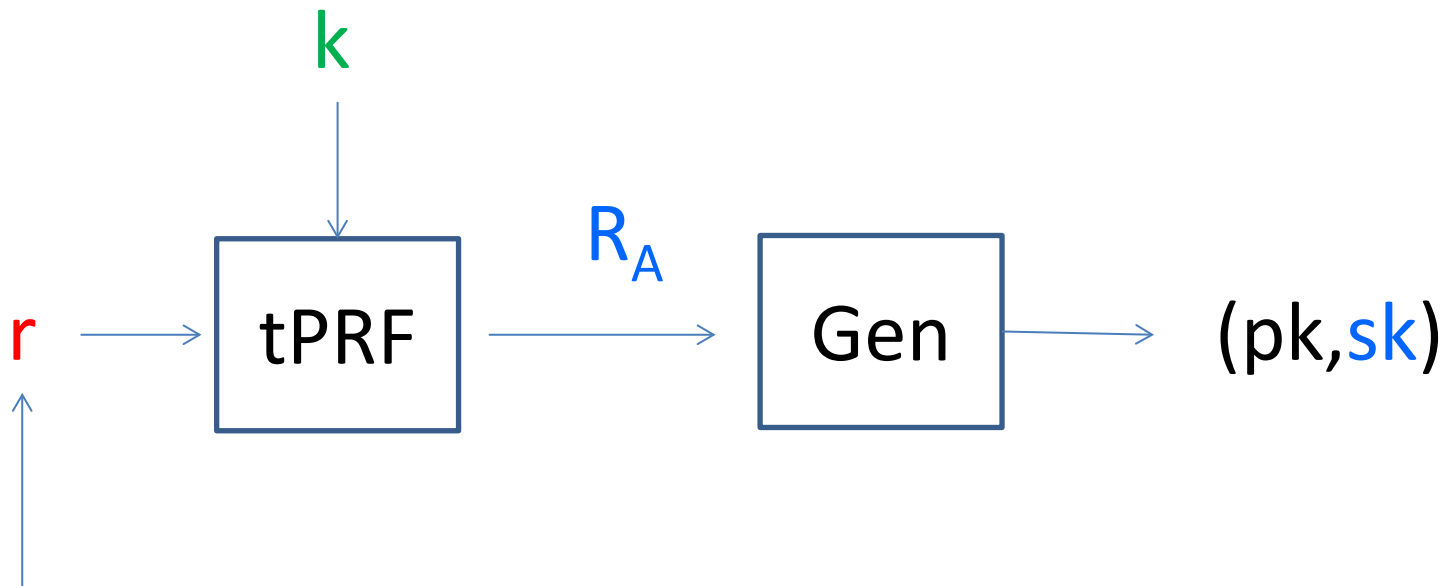
A long-term key



She keeps only r as a state

Now M cannot obtain sk

A long-term key



Because a session-state reveal query reveals only r ,
but not the long-term key k

Our 2-PASS-CK protocol

Alice has a long-term key



She chooses r randomly and computes

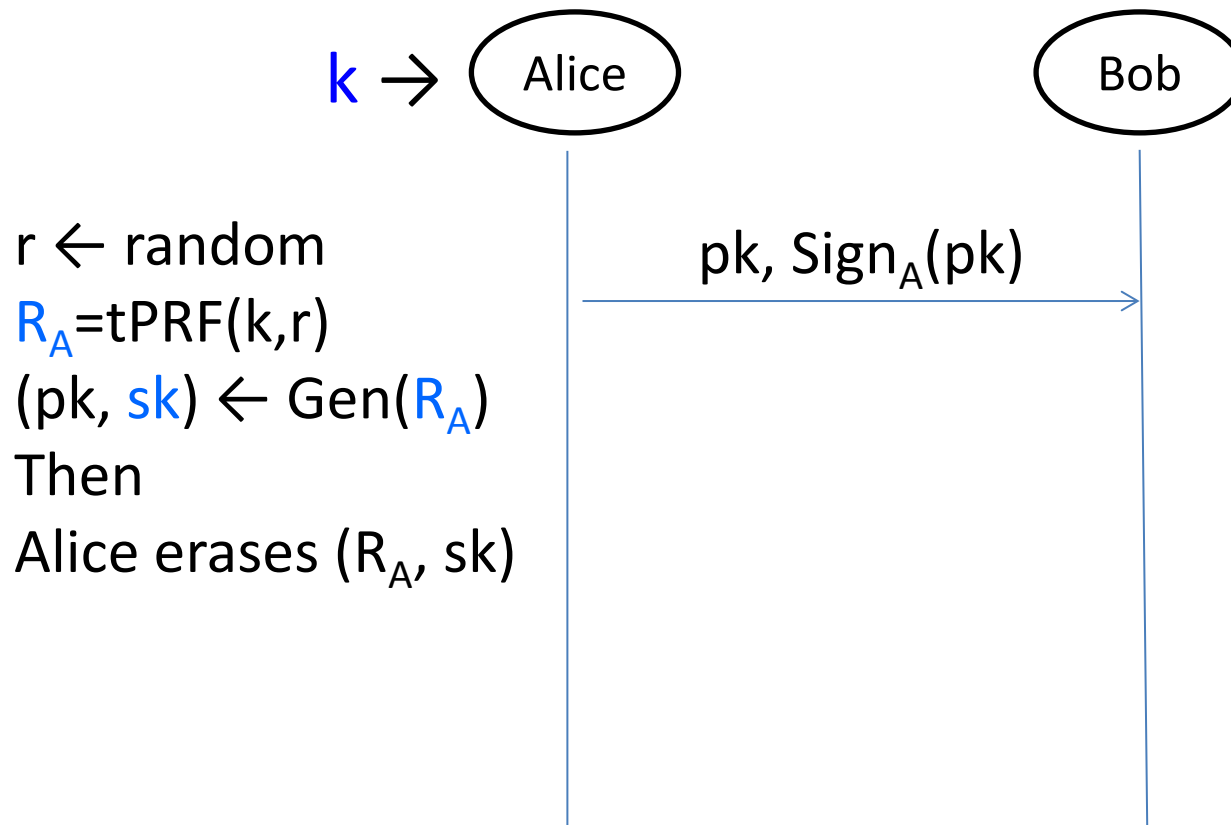
$$R_A := \text{tPRF}(k, r)$$

$$(pk, sk) := \text{Gen}(R_A)$$

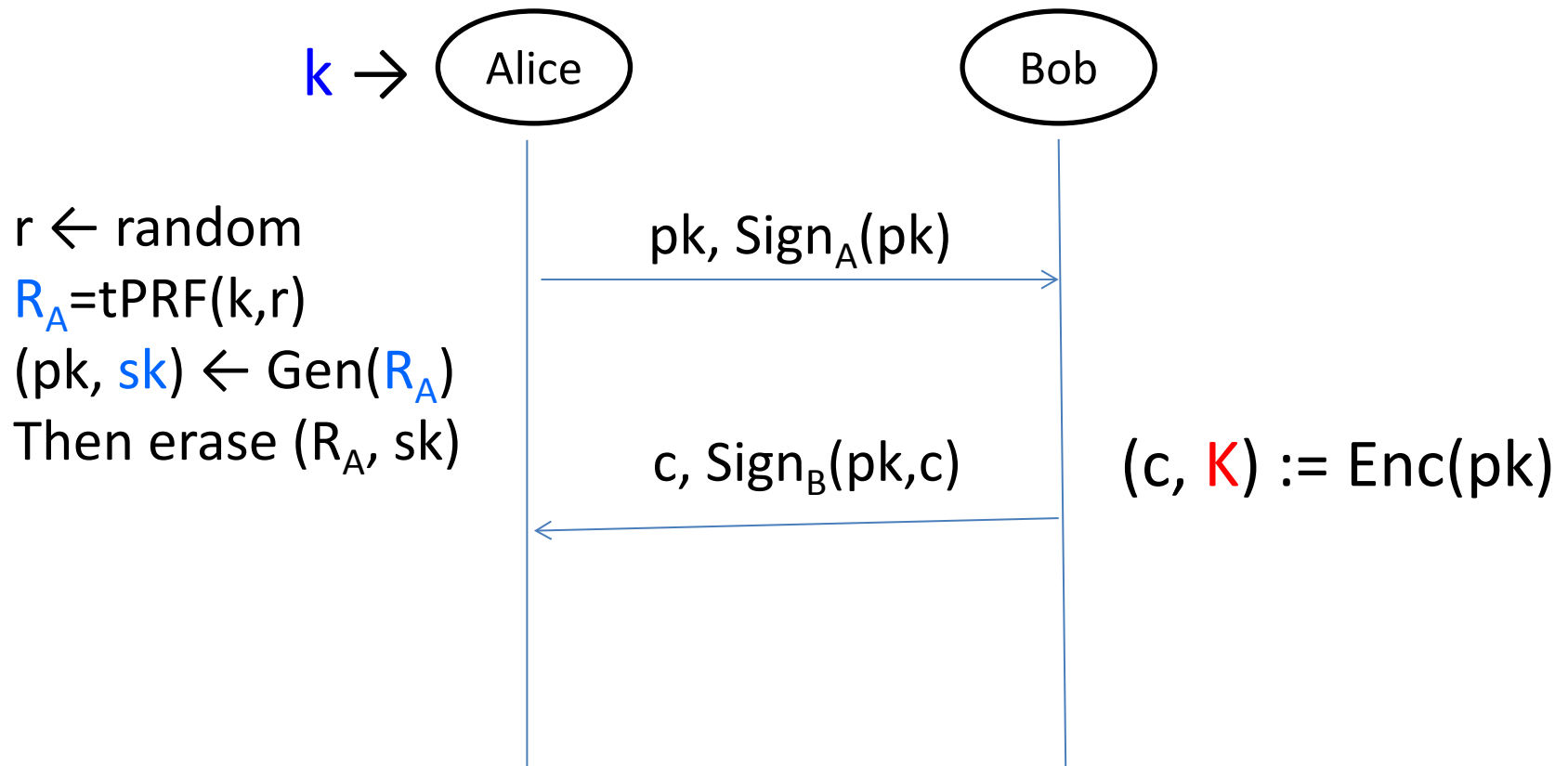
$pk, \text{Sign}_A(pk)$



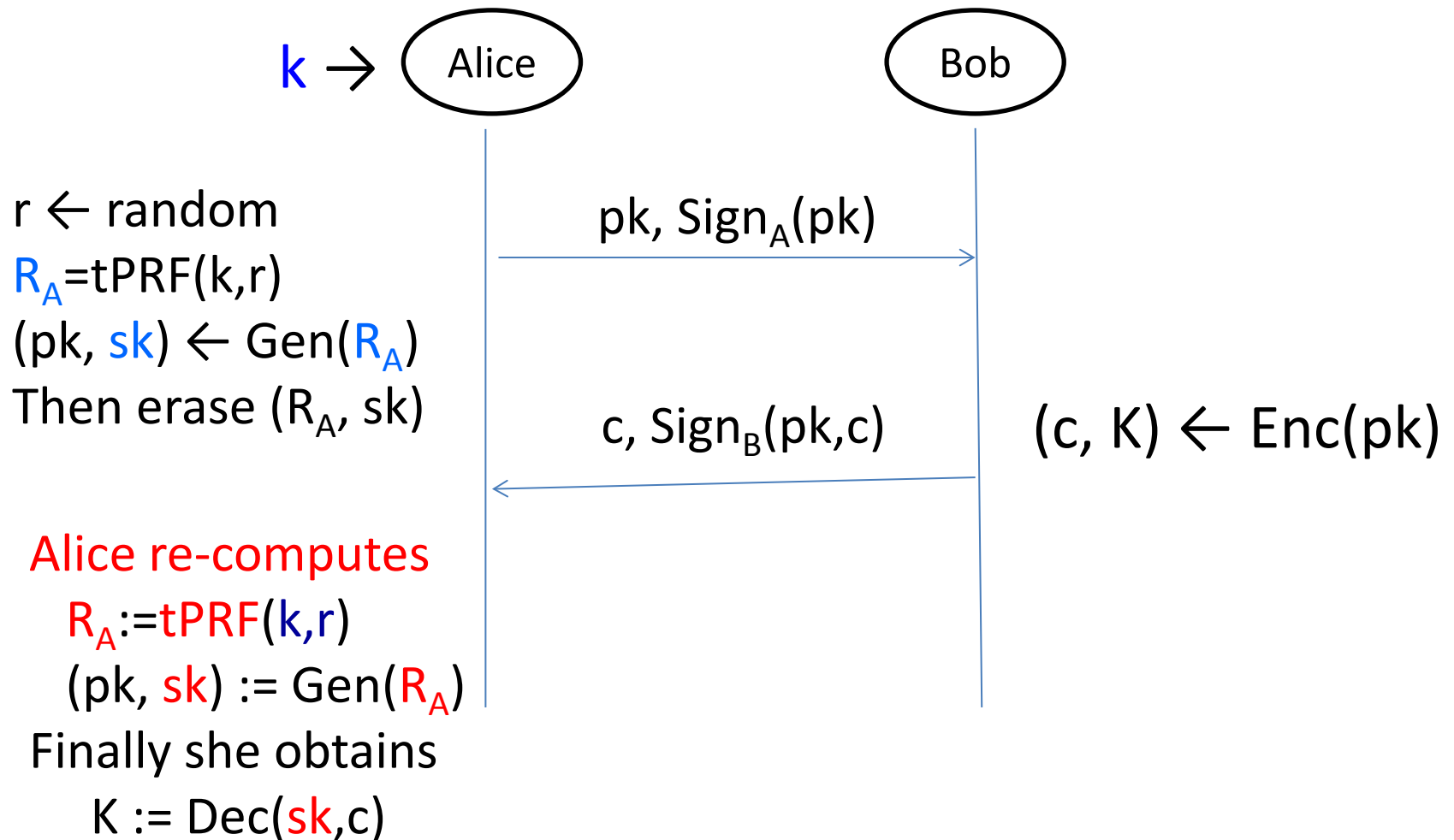
Our 2-PASS-CK protocol



Our 2-PASS-CK protocol



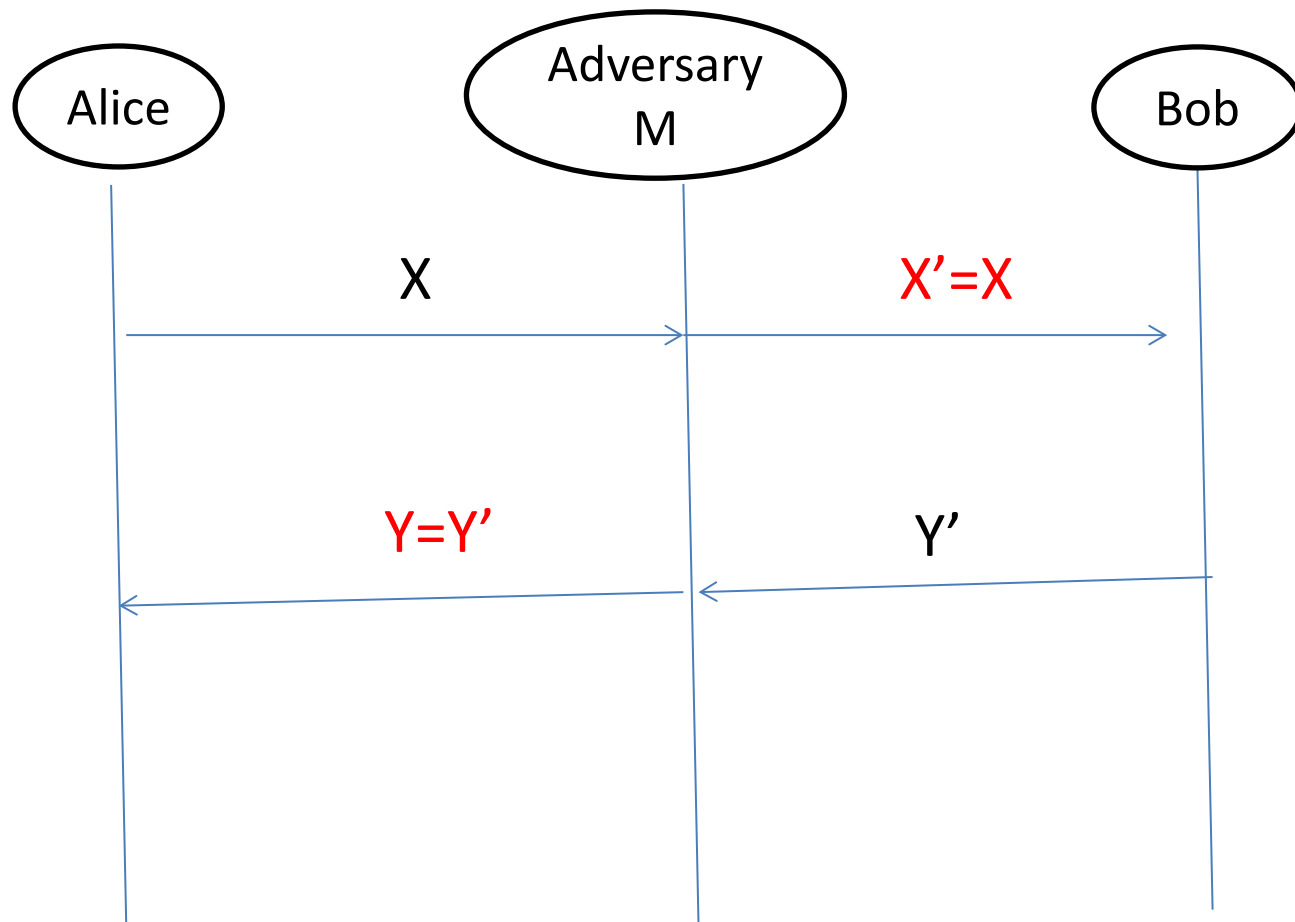
Our 2-PASS-CK protocol



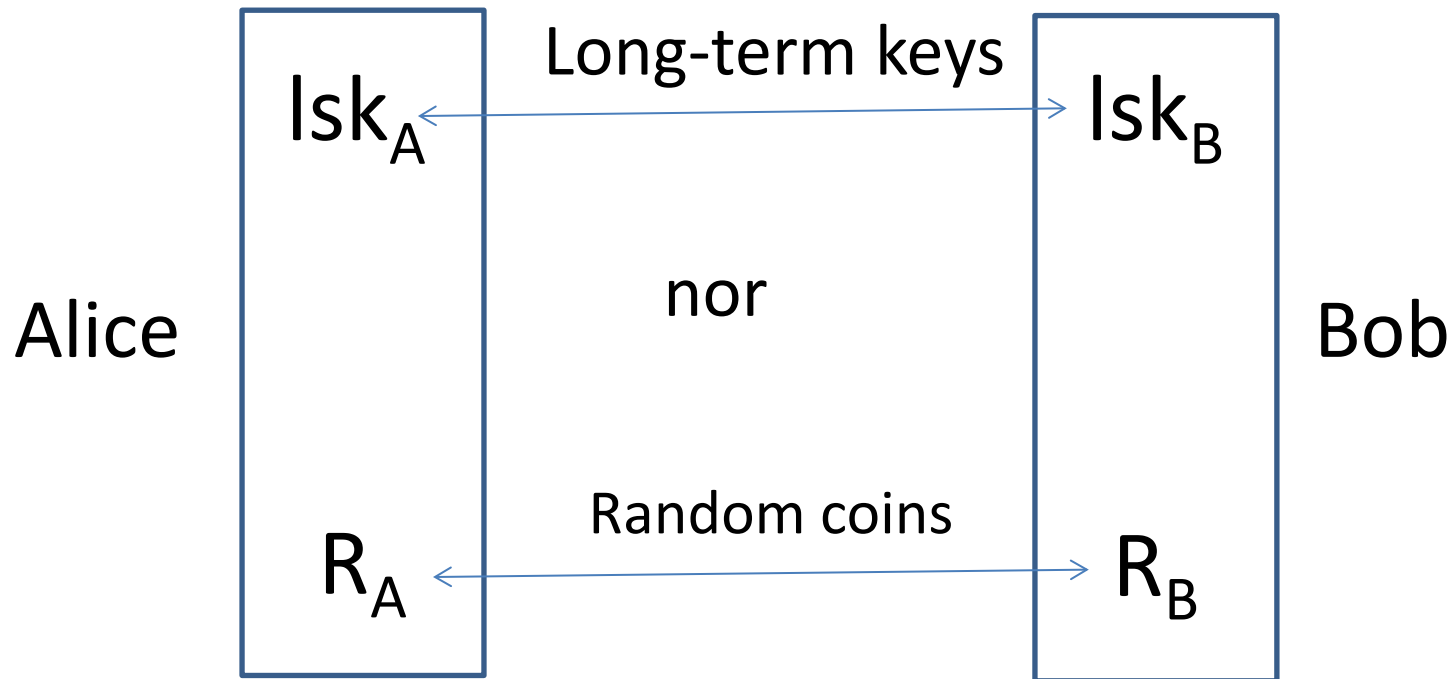
Theorem 2

- 2-PASS-CK protocol is CK-secure with wPFS
- if KEM is CPA-secure
- the signature scheme is unforgeable
- and **tPRF** is a tPRF

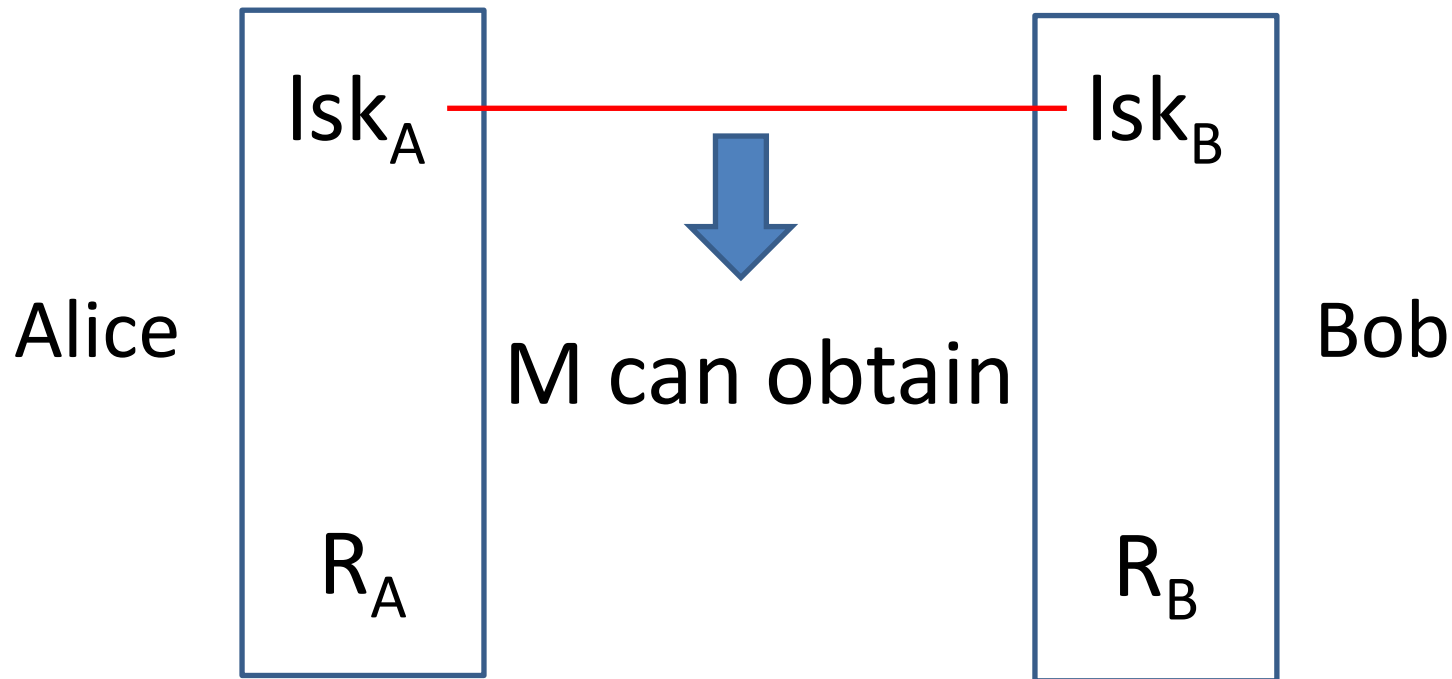
Suppose that
the test session has a matching session



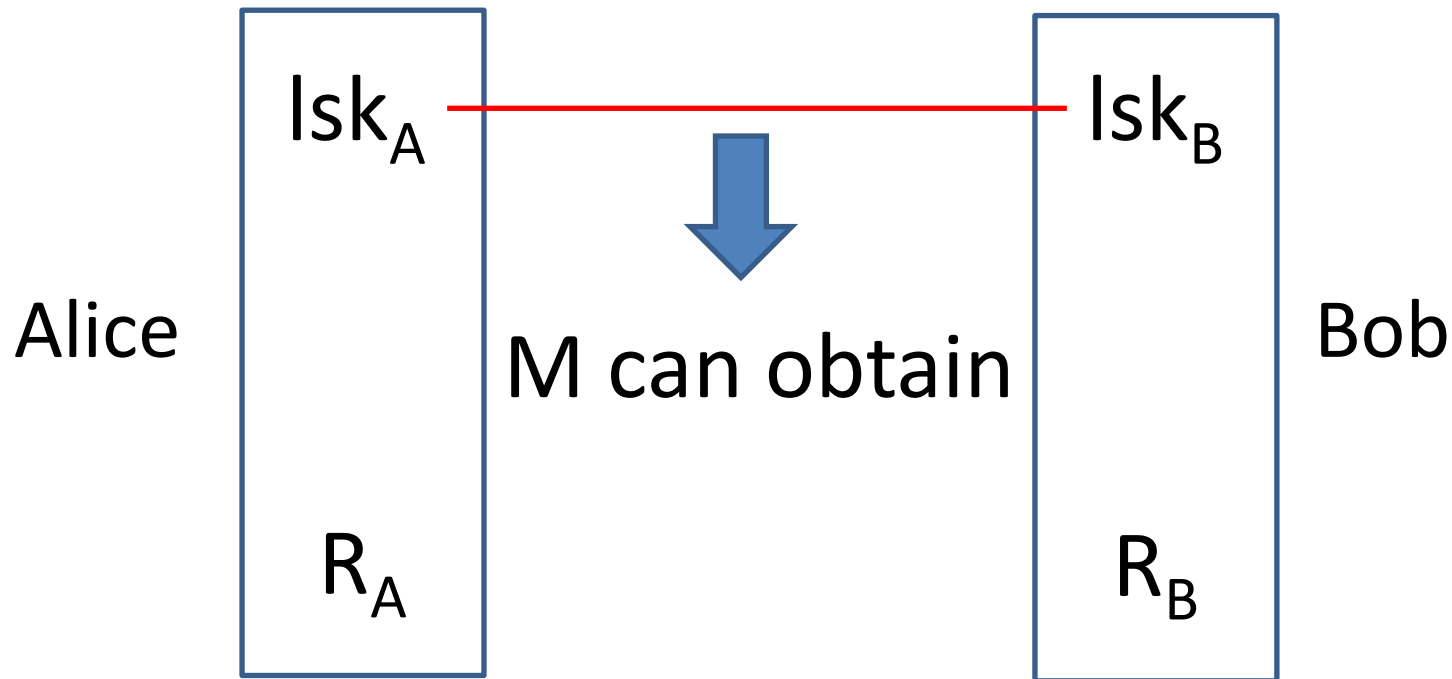
In the CK model,
nothing is revealed to M



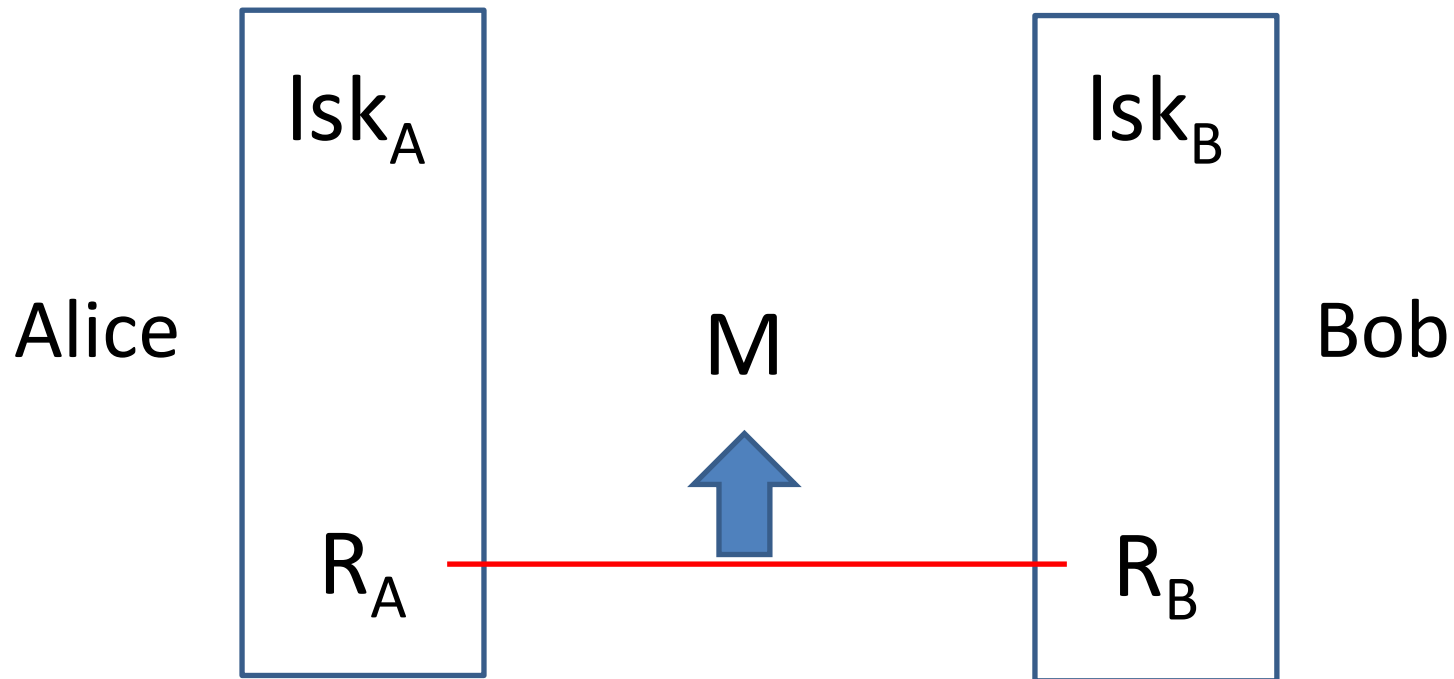
In the CK model with wPFS,



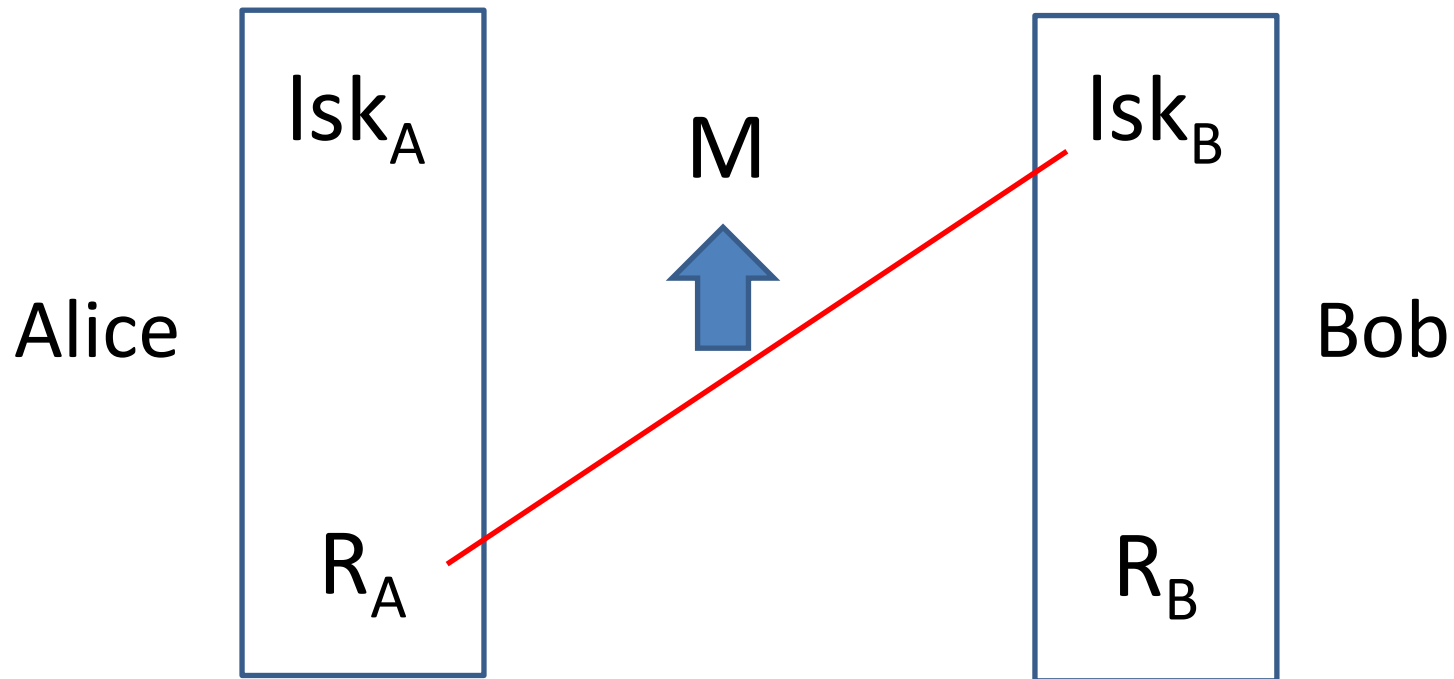
In the Extended CK (eCK) model



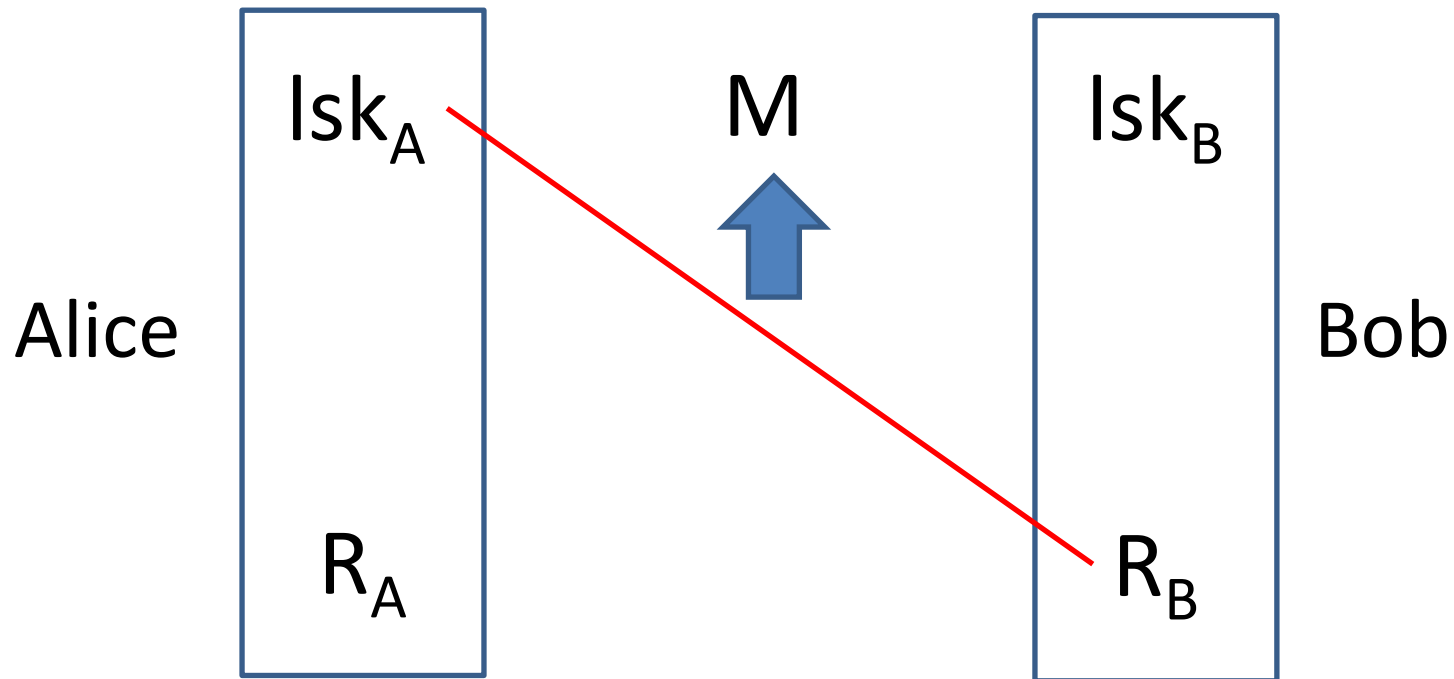
In the eCK model (2) or



In the eCK model (3) or



In the eCK model (4)
or



In our 2-PASS-CK protocol

R_A is generated by using a tPRF.

But R_B is not.

	$R_B = \text{random coins}$	R_B is by tPRF
sk is not erased		
sk is erased	CK-secure with wPFS, But not eCK-secure	

This protocol is

In our 2nd scheme,

Both R_A and R_B are generated by using a tPRF.
But sk is not erased.

	R_B = random coins	R_B is by tPRF
sk is not erased		eCK-secure But not CK-secure
sk is erased	CK-secure with wPFS, But not eCK-secure	



This protocol is

In our 3rd scheme

Both R_A and R_B are generated by using a tPRF and sk is erased.

	R_B is not tPRF	R_B is tPRF
sk is not erased		eCK-secure, but not CK-secure
sk is erased	CK-secure with wPFS, but not eCK-secure	CK-secure with wPFS and eCK-secure

This protocol is

Our results

- Make it clear that
- there exists a clear separation
- between CK-security and eCK-security

Summary (1)

	round	wPFS	By using
Fujioka et al.	2-pass	○	CCA-KEM
We constructed	2-pass	○	CPA-KEM

Our assumption is weaker
than Fujioka et al.

Summary (2)

Our	security
1 st scheme is	CK-secure
2 nd scheme is	eCK-secure
3 rd scheme is	Both CK and eCK-secure

Thank you !