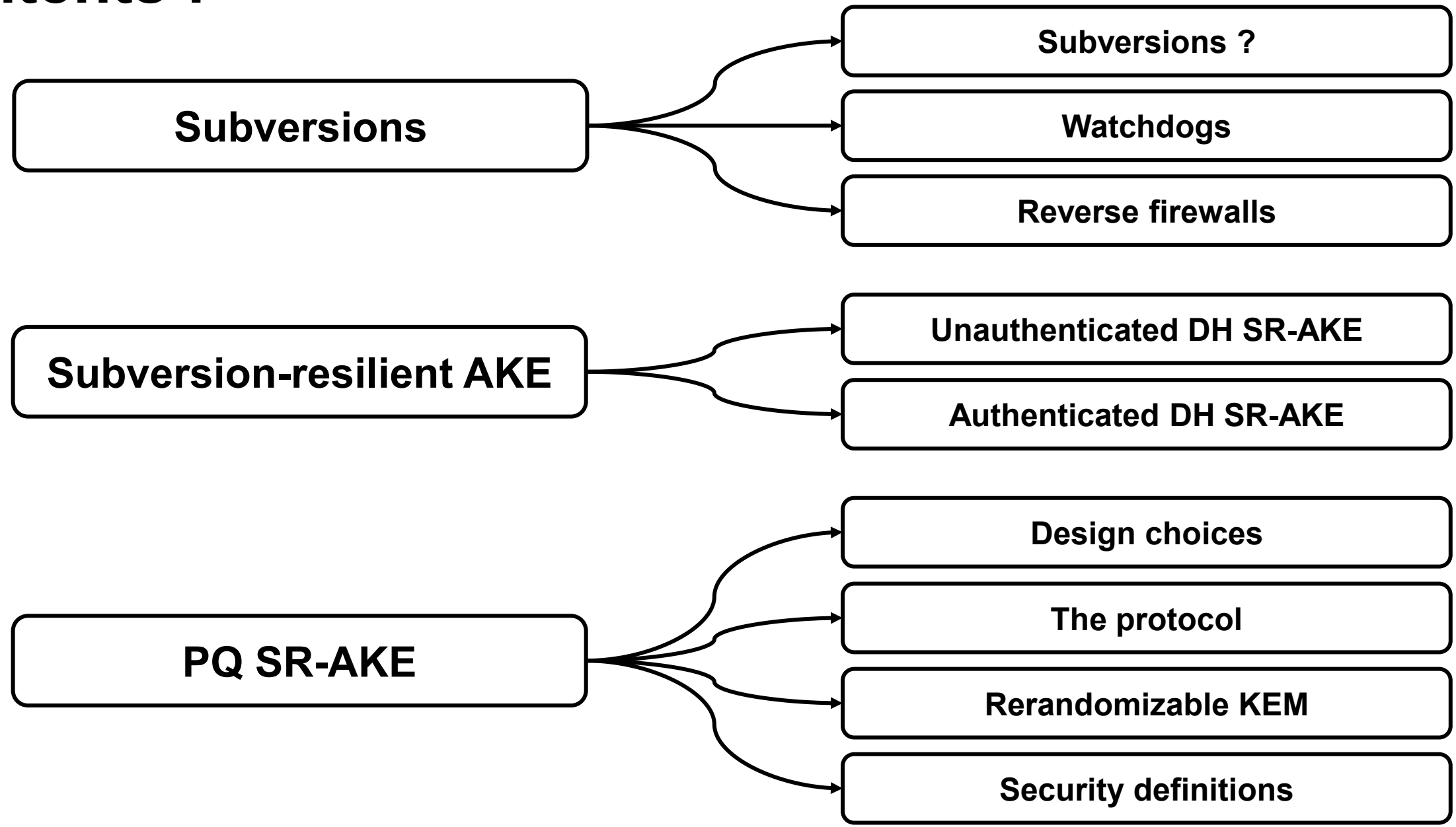
Journée 5 - Axe 2 - PEPR-PQ TLS

PQ SR-AKE : Subversion-resilient key-exchange in the post-quantum world

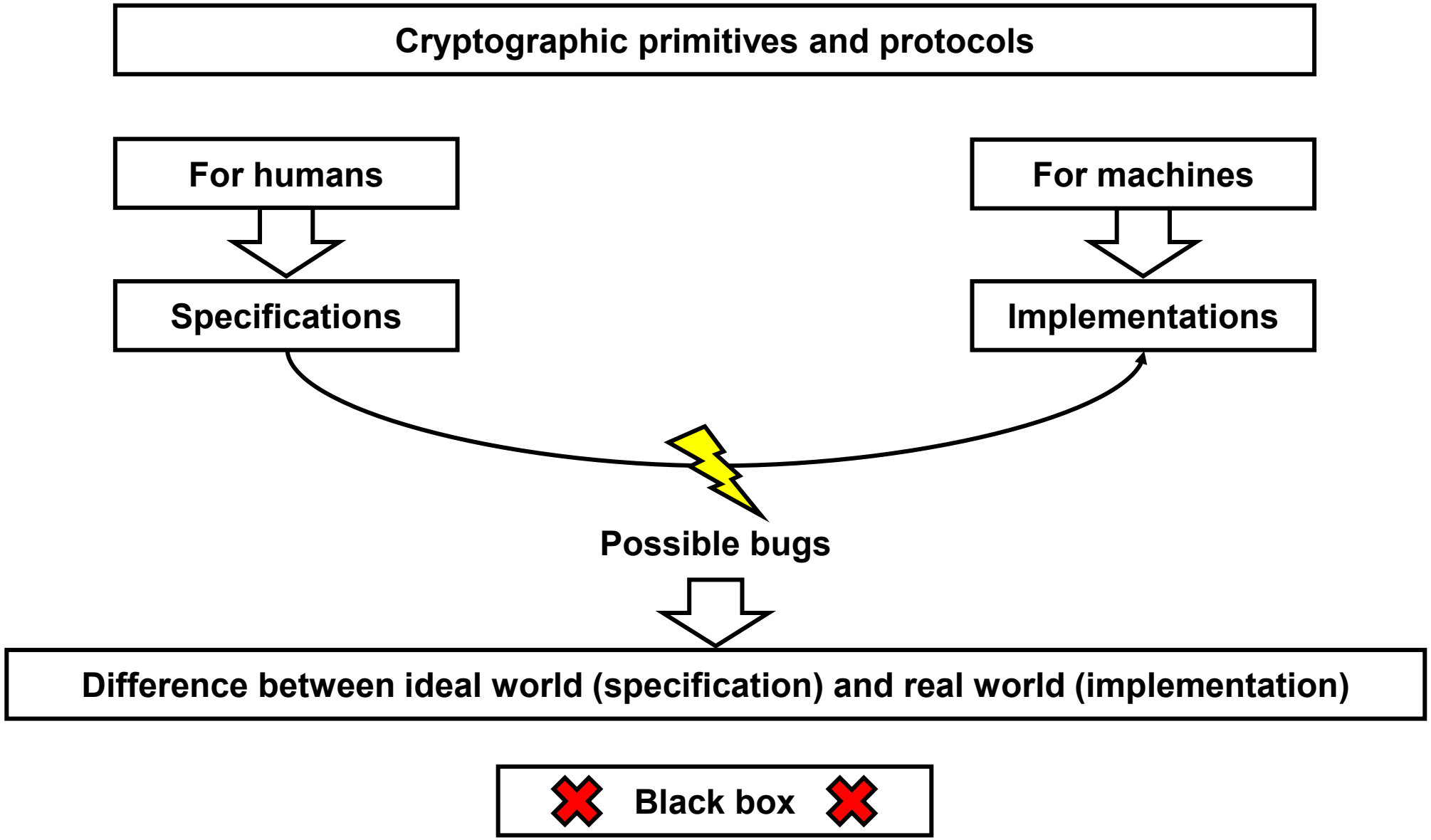
Kevin DUVERGER, Pierre-Alain FOUQUE, Charlie JACOMME, Guilhem NIOT, Cristina ONETE

Speaker – Kevin DUVERGER **10 / 07 / 2025**

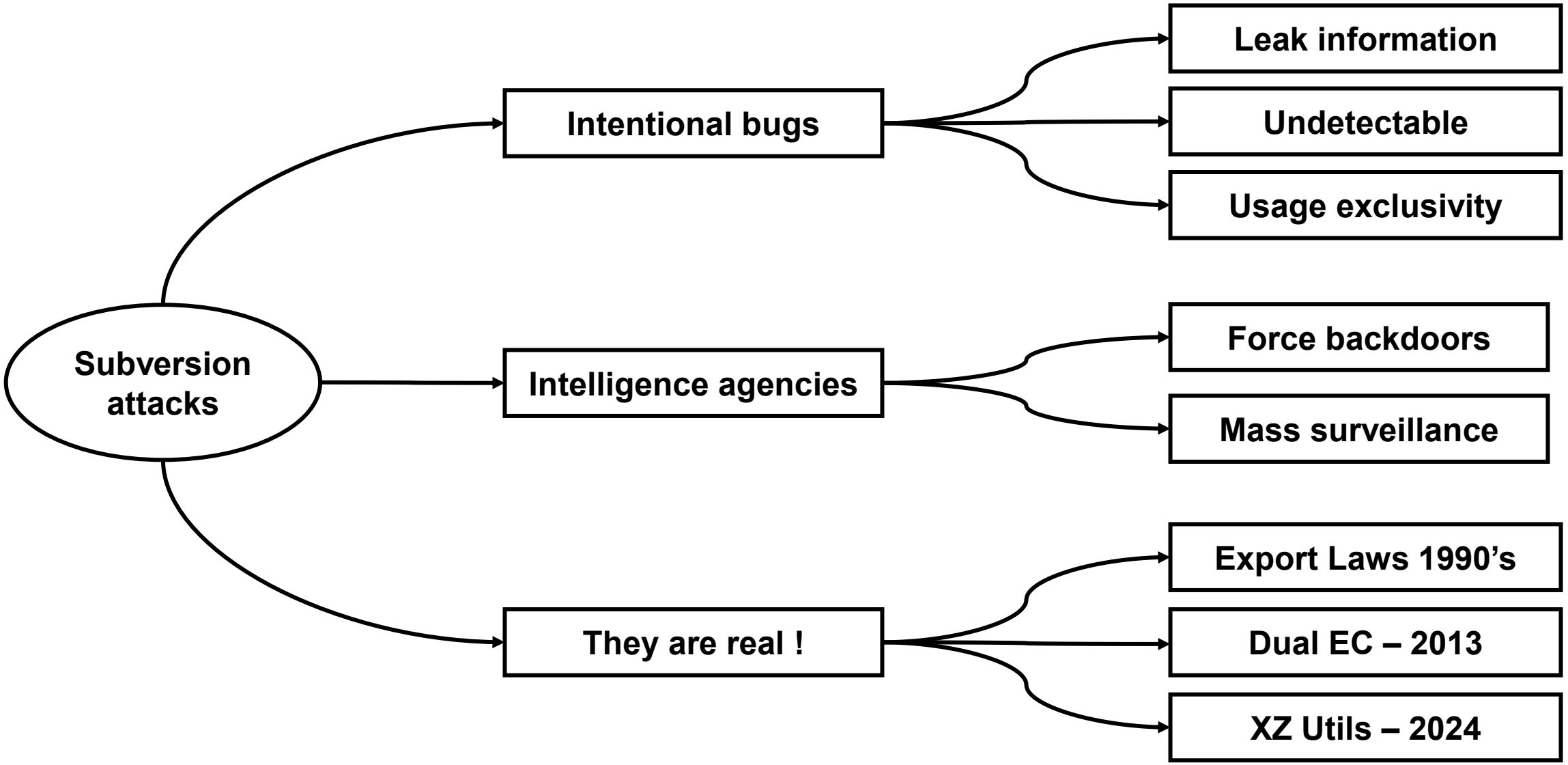
Contents :


Subversions
Subversions ?
Watchdogs
Reverse firewalls
Subversion-resilient AKE
Unauthenticated DH SR-AKE
Authenticated DH SR-AKE
PQ SR-AKE
Design choices
The protocol
Rerandomizable KEM
Security definitions

Introduction :



Introduction :



Introduction :

IND-CPA ROR – KEM
$(pk, sk) \leftarrow^{\$} \mathrm{KeyGen}(1^{\lambda})$ $(ct, K_0) \leftarrow^{\$} \mathrm{Encaps}(pk)$ $K_1 \leftarrow^{\$} \mathcal{K}$ $b \leftarrow^{\$} \{0,1\}$ $d \leftarrow \mathcal{A}(pk, ct, K_b)$
$\mathcal{A}$ **wins iff** $b = d$

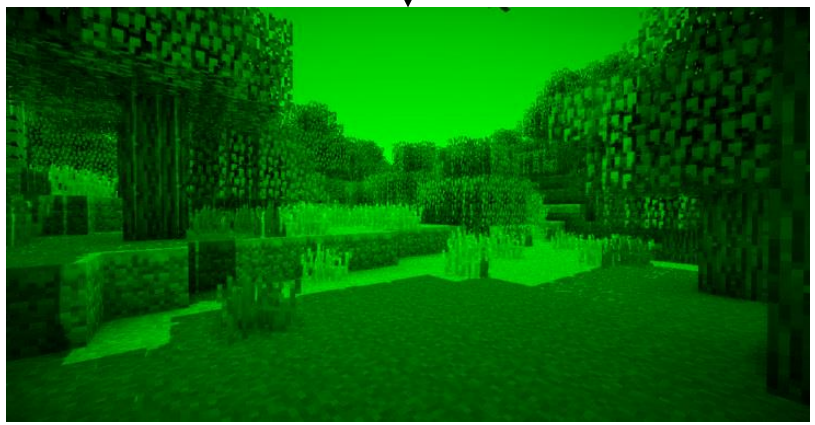
EUF-CMA – SIGN
$(pk, sk) \leftarrow^{\$} \mathrm{KeyGen}(1^{\lambda})$ $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathrm{oSign}(.)}(pk)$
$\mathcal{A}$ **wins iff** $\mathrm{Verify}(pk, \sigma^*, m^*) = 1 \wedge m^* \notin \mathcal{L}$

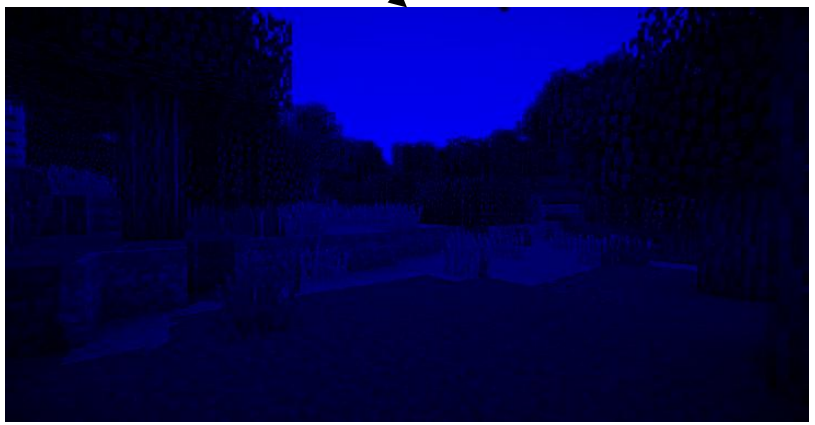
$\mathrm{oSign}(m)$
$\sigma \leftarrow^{\$} \mathrm{Sign}(sk, m)$ $\mathcal{L} \leftarrow \mathcal{L} \cup \{m\}$ return σ

Algorithms are idealized !

Need to consider implementation vs specification in models !

Subversions – Attacks :





Subversions : similar to steganography on cryptographic elements

Subversions – Attacks :

Starting point : subliminal channels, Gustavus J. Simmons, 1980's

Attack : needs $\mathrm{GetRandom}(\sigma, sk) \rightarrow r$

Examples : Ong-Schnorr-Shamir / ElGamal

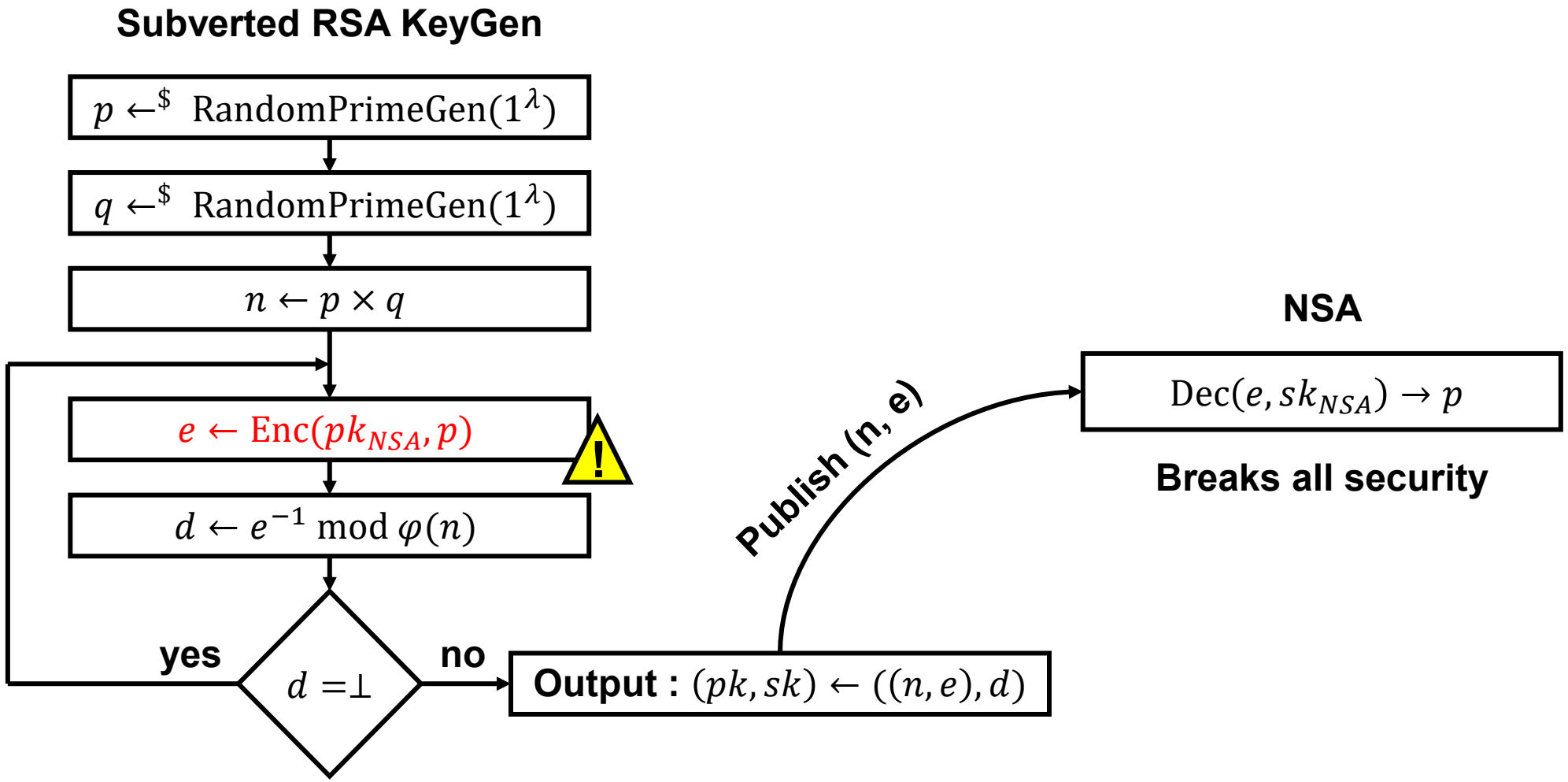
$\sigma = \mathrm{Sign}(sk_R, m, m^*)$

OK

$\mathrm{GetRandom}(\sigma, sk_R) \rightarrow m^*$

Subversions – Attacks :

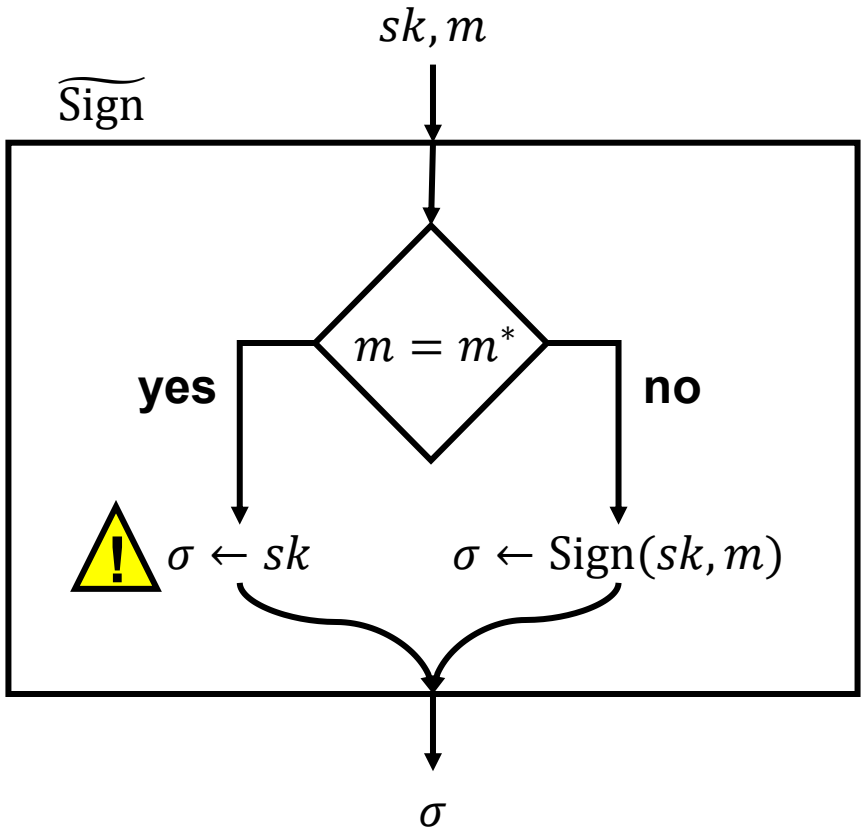
Kleptography, Young and Yung, 1990's



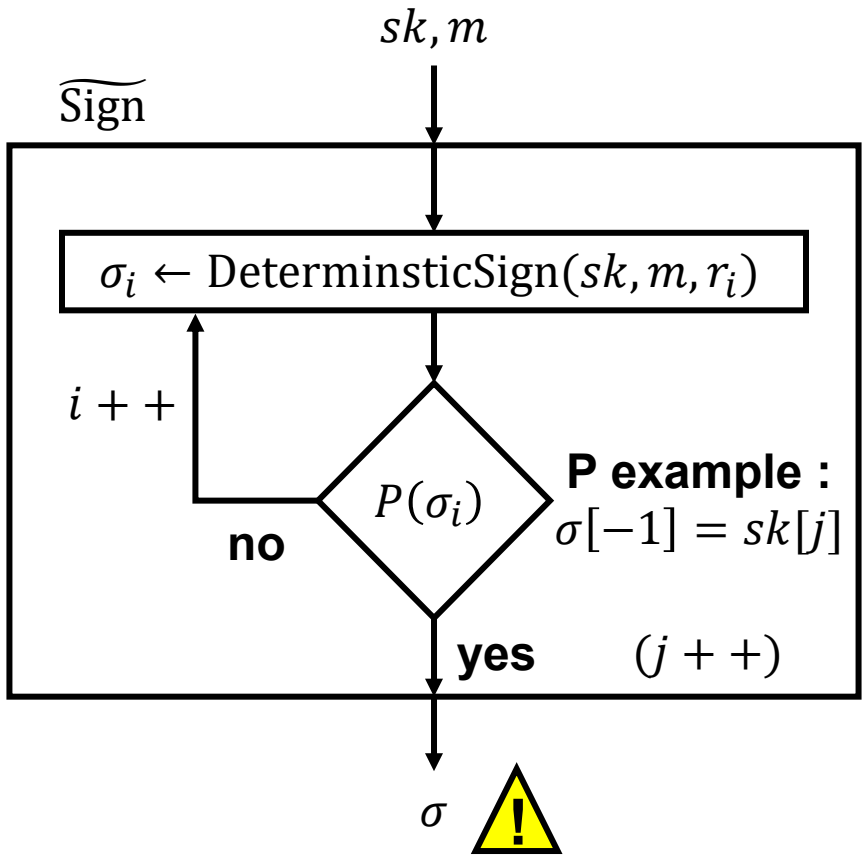
Subversions – Attacks :

Algorithm Substitution Attacks (2014 – Now)

Input trigger



Rejection sampling

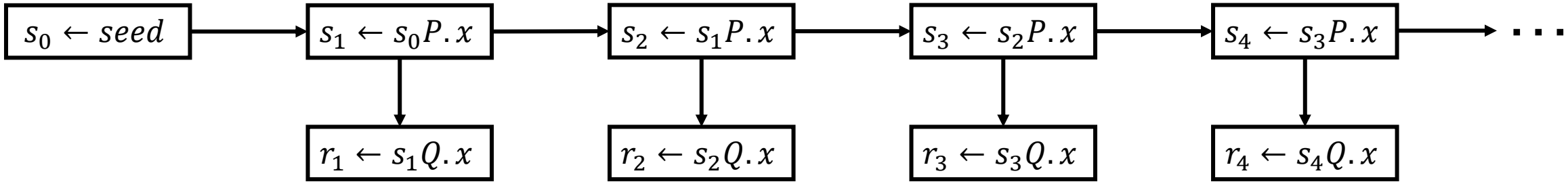


Exclusivity : encrypt, Undetectability : for later

Subversions – Real world attacks :

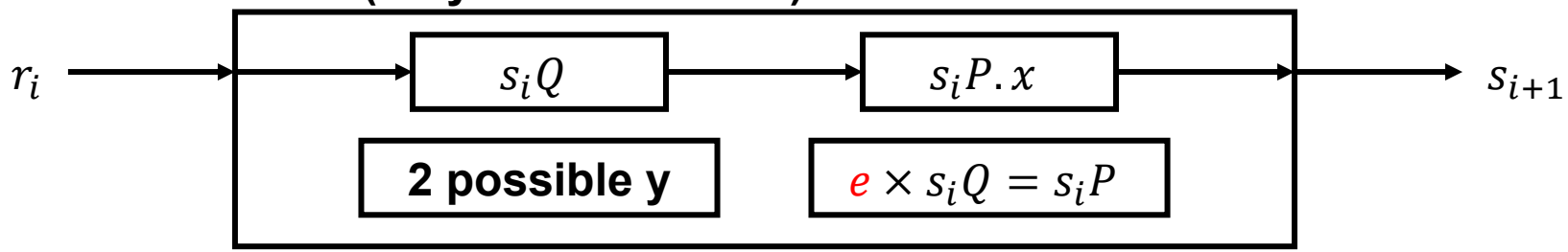
Edward Snowden – Dual EC DRBG – 2013

Global parameters : P, Q



What if $P = eQ$ **?**

NSA (only one to know e**)**

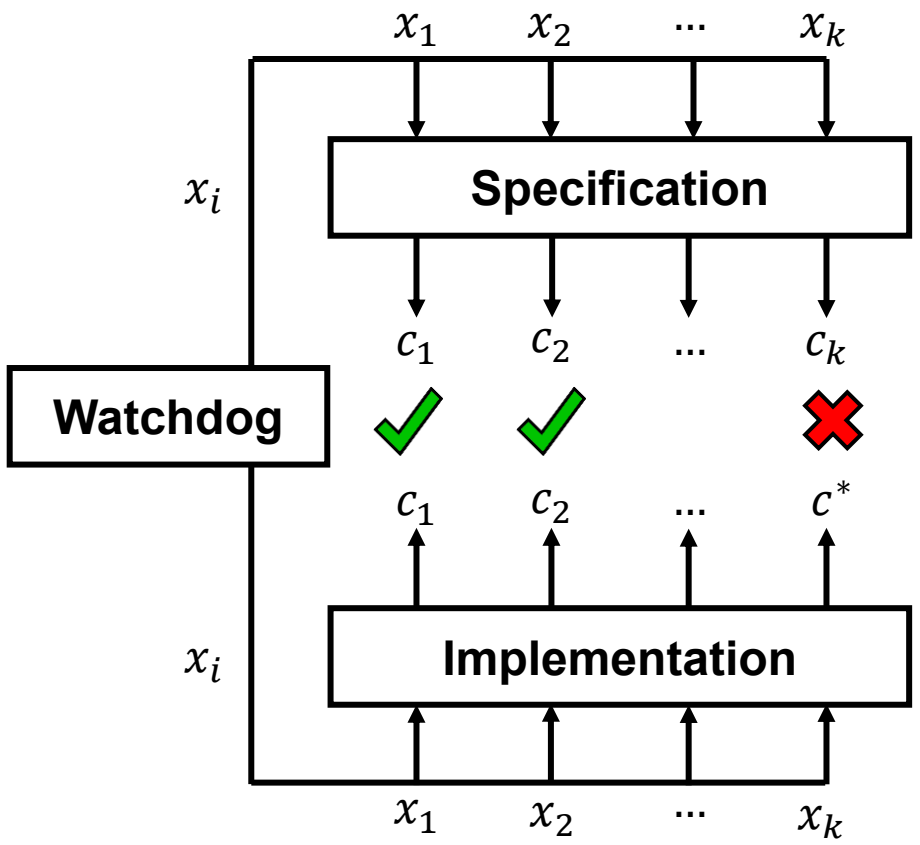


Subversions – Watchdogs :

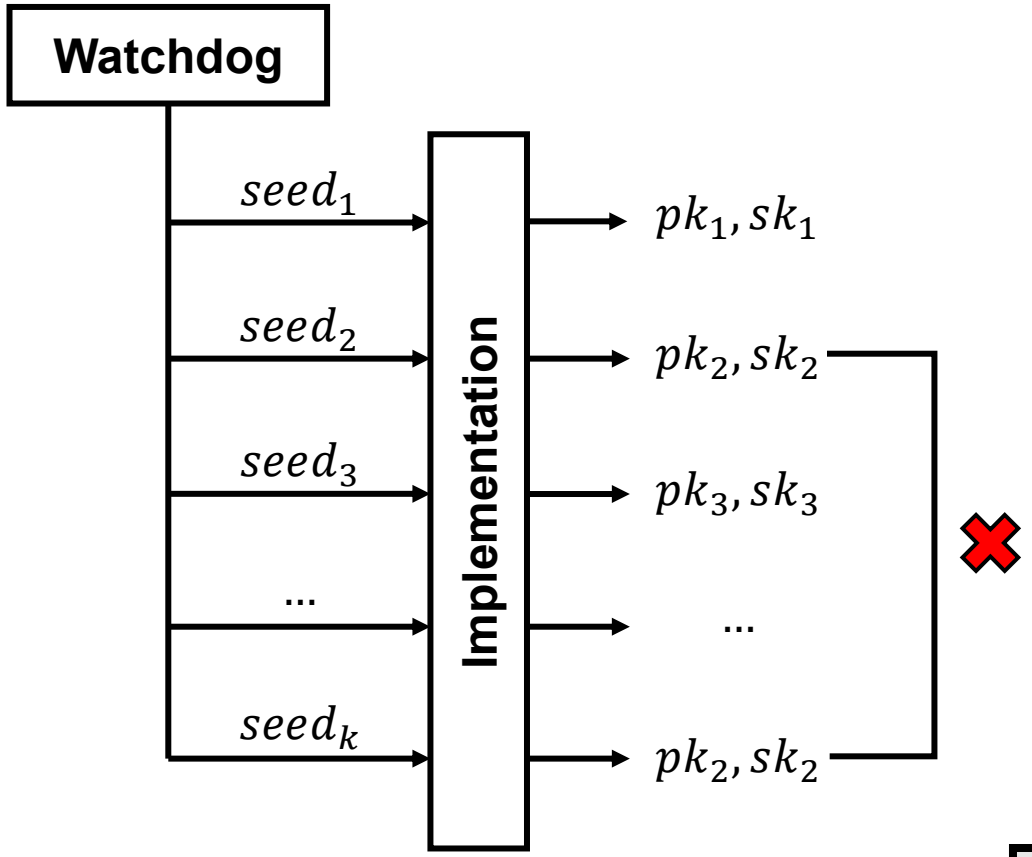
Offline watchdogs = test before production – Blackbox testing

Deterministic watchdog (RSA encryption)

$x_i = (pk_i, m_i) \leftarrow^{\$} (\mathcal{PK} \times \mathcal{M})$



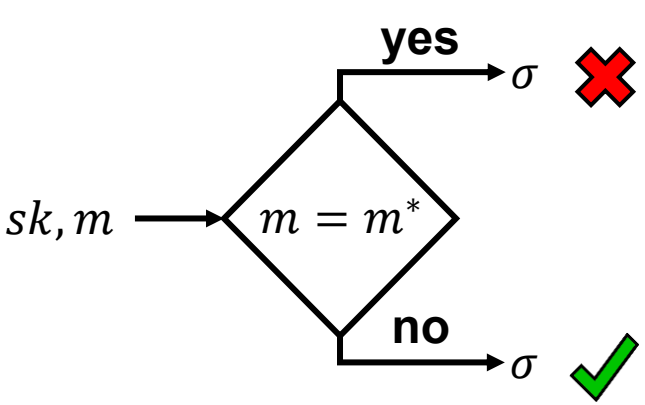
Probabilistic (RSA keygen)



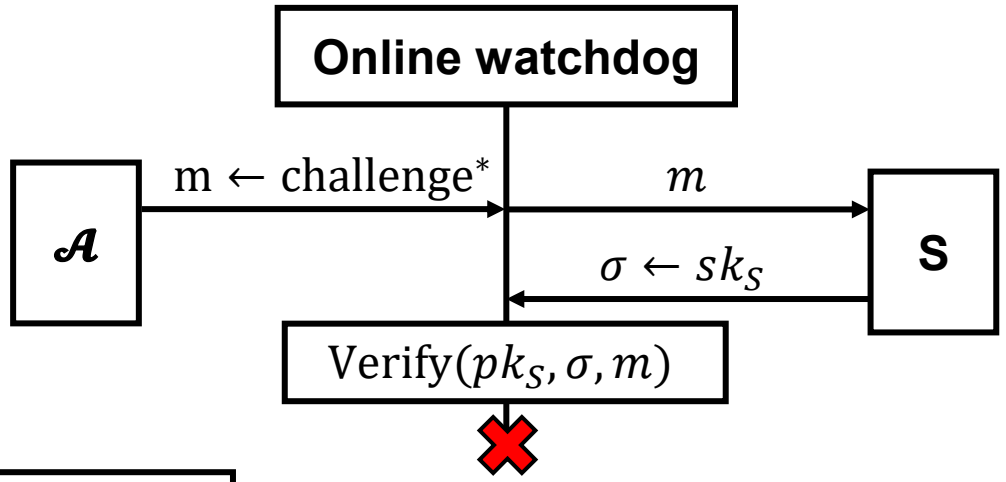
Subversions – Watchdogs :

Online watchdog = test always – Blackbox testing

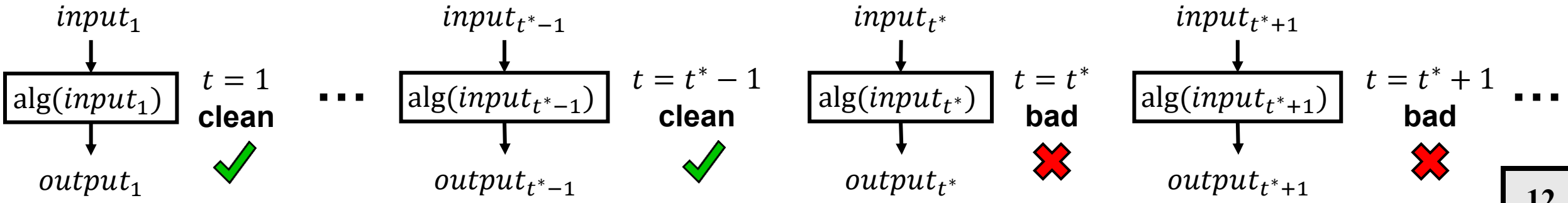
Input trigger (signature)



$|\mathcal{M}| = \infty$
$\downarrow$
$\mathrm{P}(\mathrm{detect}) = 0$



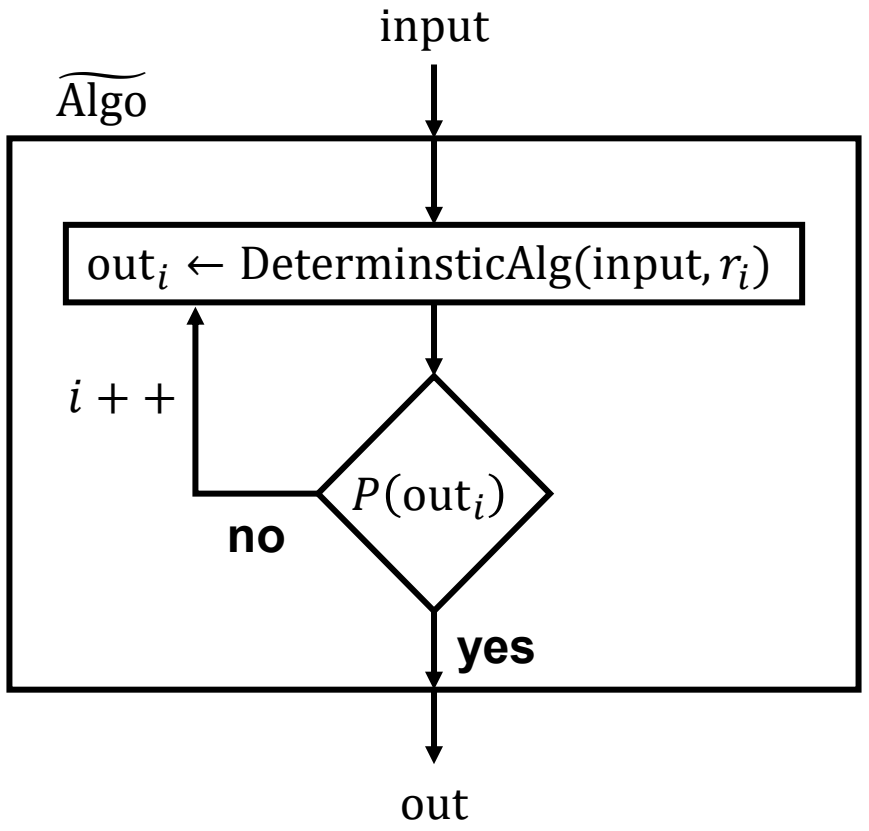
Time bomb (can detect too)



Subversions – Watchdogs :

Blackbox testing NOT SUFFICIENT – Fine grain testing

Rejection sampling



Undetectable for any blackbox test

Solution (KeyGen example)

Split-program model	RandGen	dKeyGen
Trusted amalgamation	**Trusted ∘**	

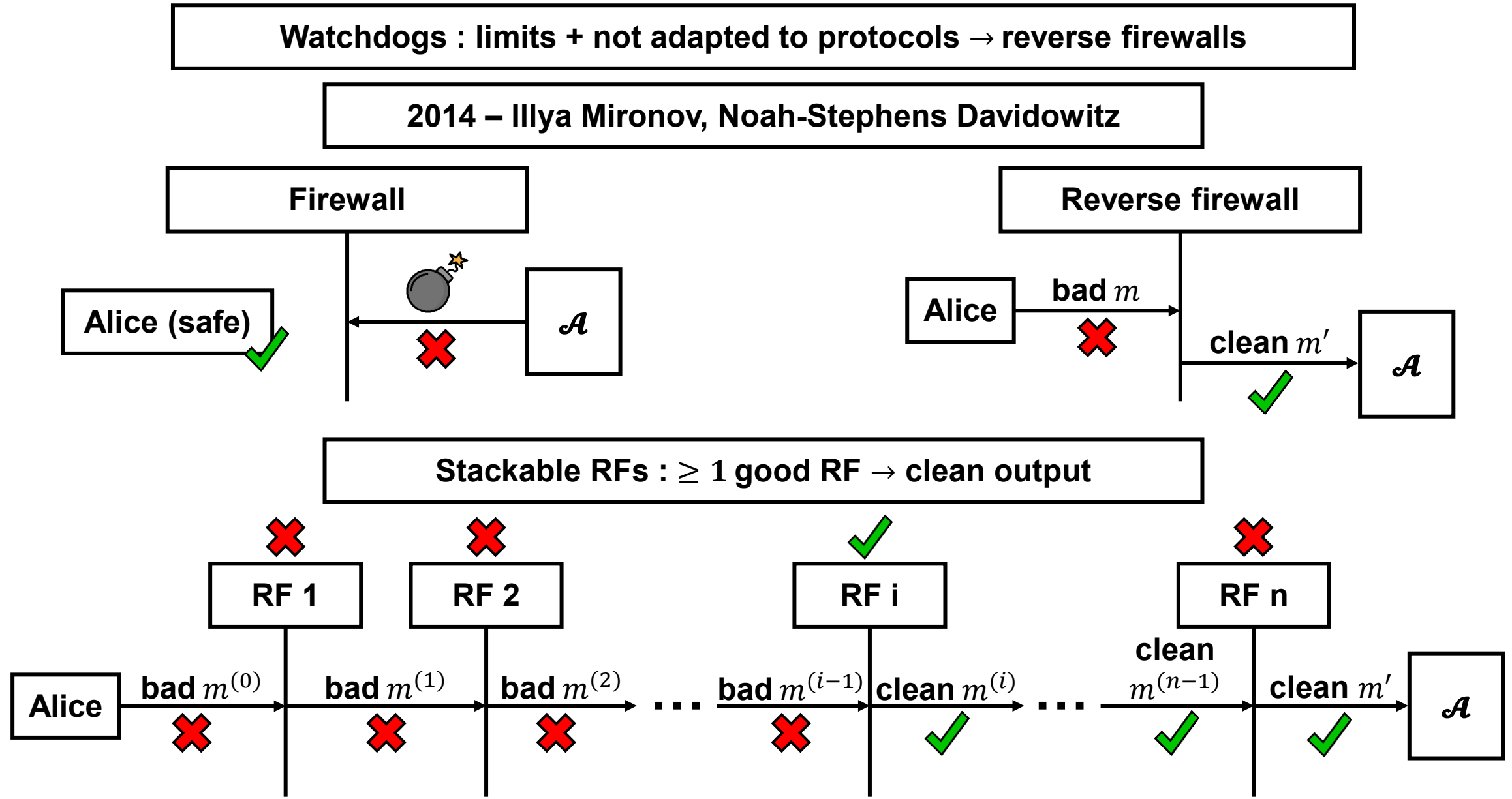
$$\text{KeyGen} = \text{dKeyGen} \circ \text{RandGen} = \text{dKeyGen}(RandGen)$$

Protected KeyGen (double splitting)

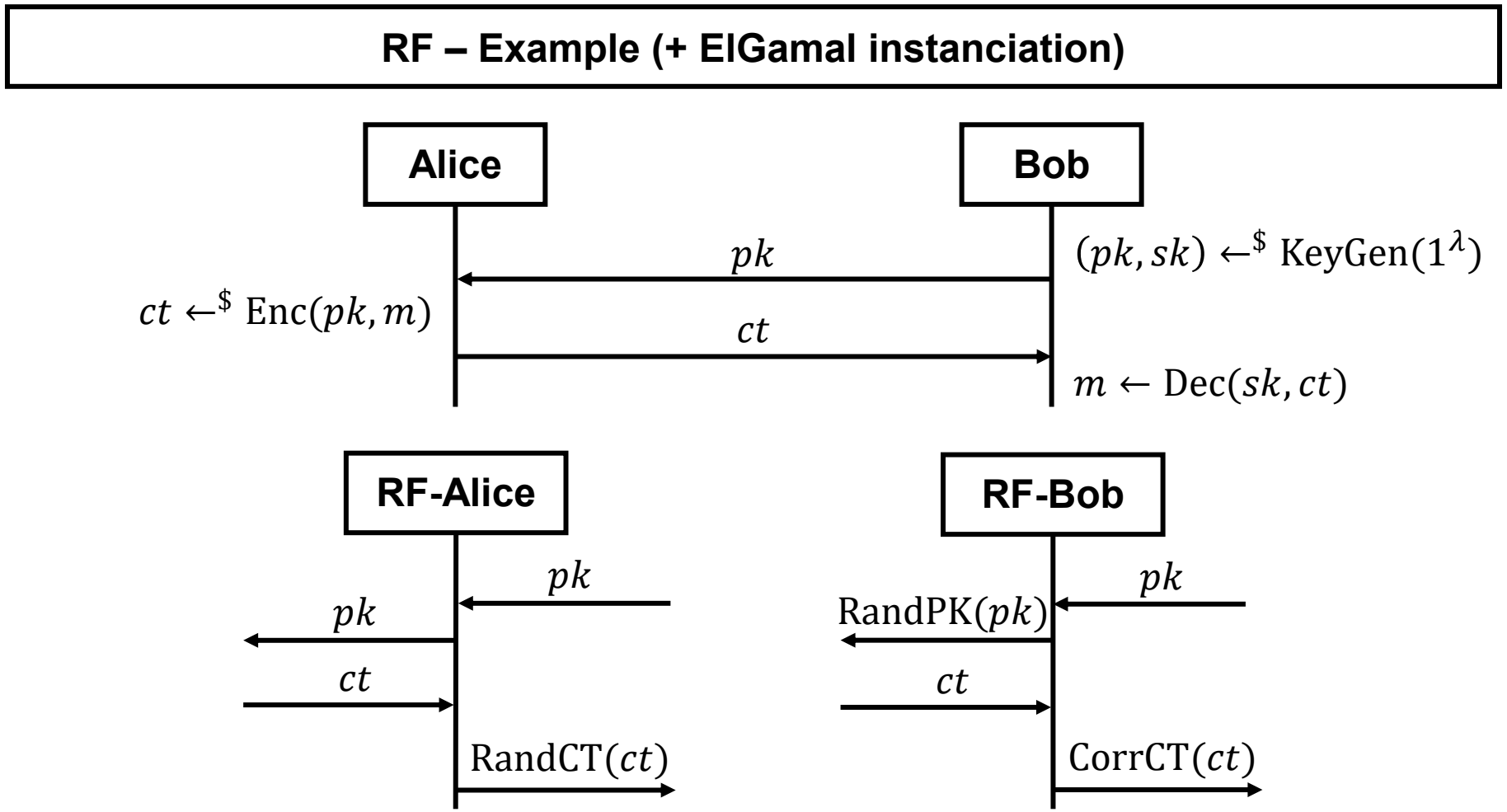
RG_0	RG_1	RandExtract	dKeyGen

$$KeyGen() = dKeyGen\big(RandExtract(RG_0(), RG_1())\big)$$

Subversions – Reverse firewalls :



Subversions – Reverse firewalls :



$\text{KeyGen}(1^{\lambda}) : (x, g^{x})$ $\quad \text{Enc}(pk, m) : (g^{r_0}, m \times pk^{r_0})$ $\quad \text{Dec}(sk, ct) : ct_1 / ct_0^{sk}$

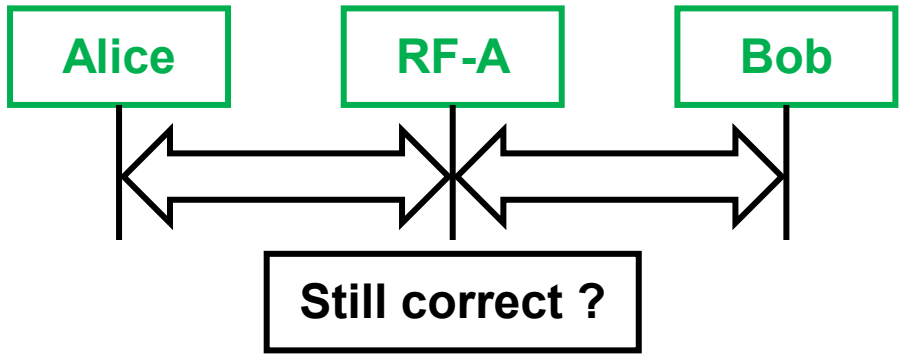
$\text{RandCT}(ct) : (g^{r_1} \times ct_0, pk^{r_1} \times ct_1)$ $\quad \text{RandPK}(pk) : pk^{r_2}$ $\quad \text{CorrCT}(ct) : (ct_0^{r_2}, ct_1)$

Subversions – Reverse firewalls :

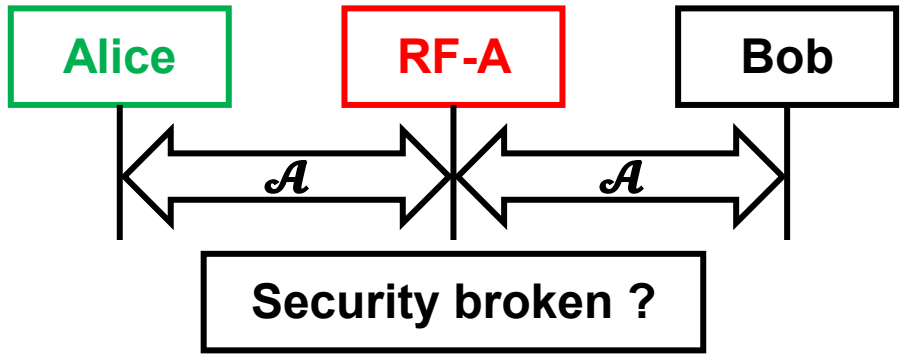
RF – Properties

■ : Honest, ■ : Malicious, ■ : Subverted

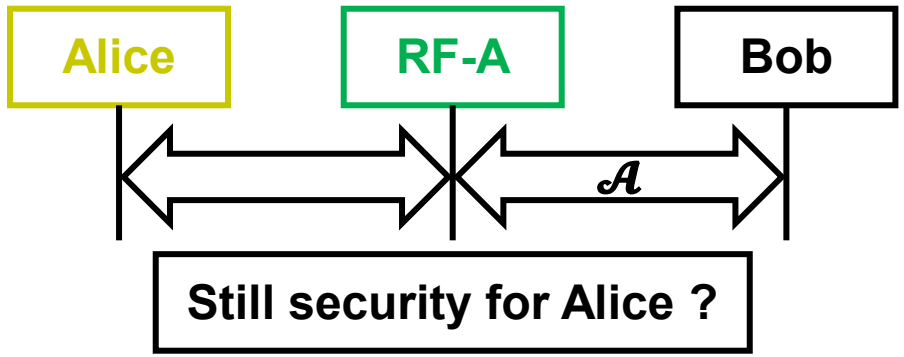
Functionality maintaining



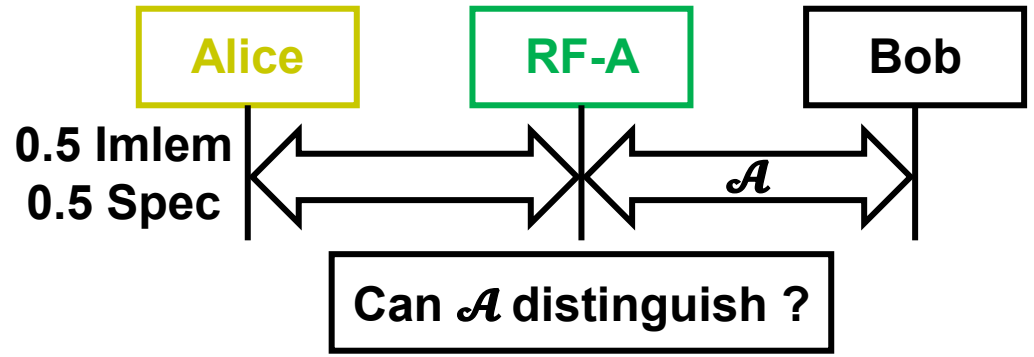
RF not trustworthy



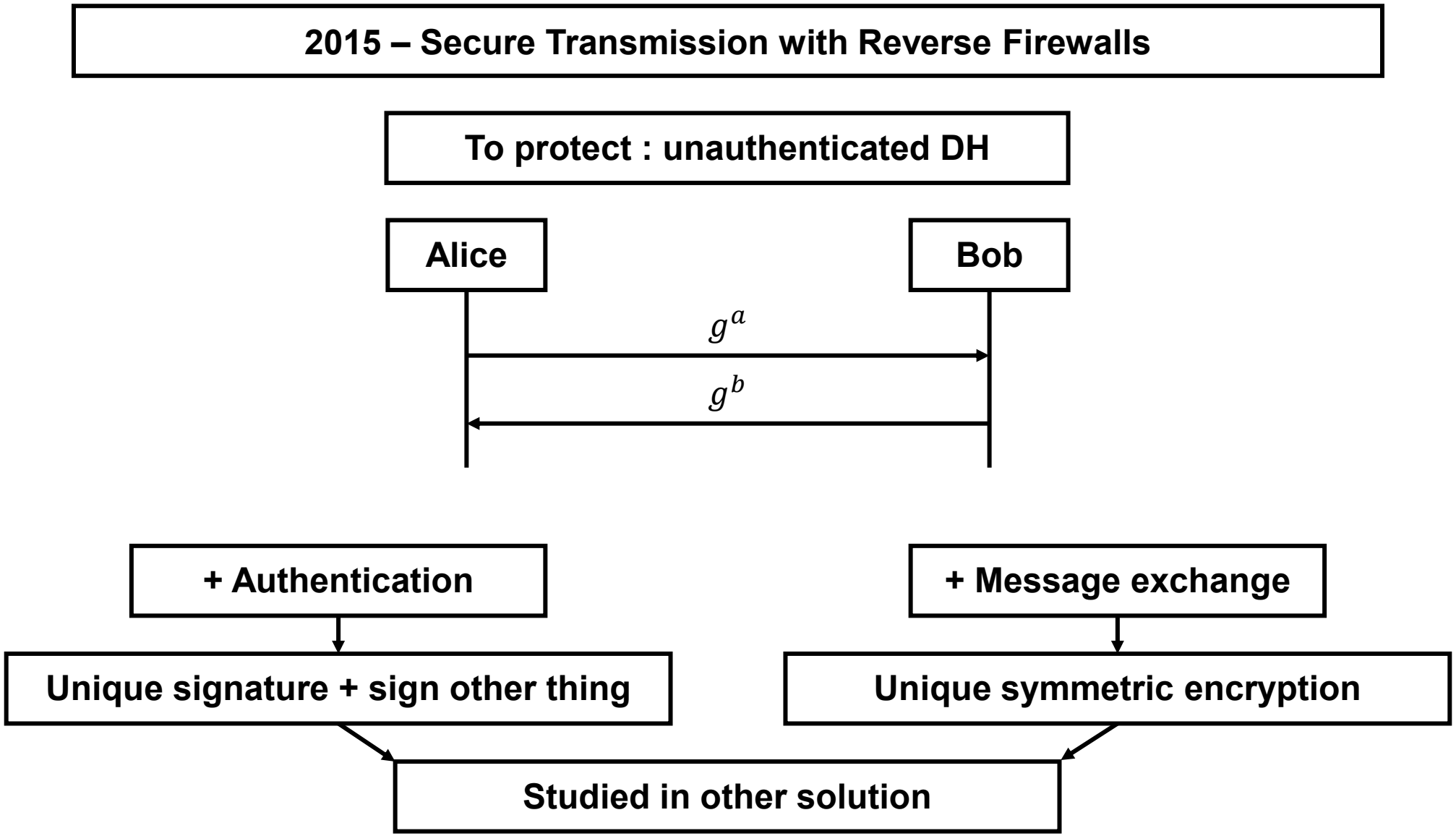
Security preserving



Exfiltration-resilience



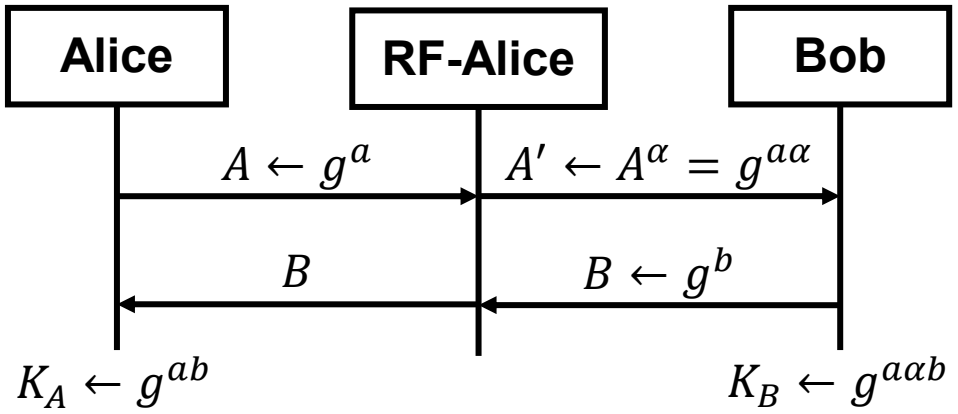
Subversion-resilient AKE – Example 1 :

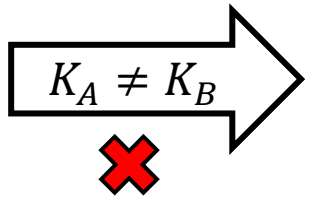


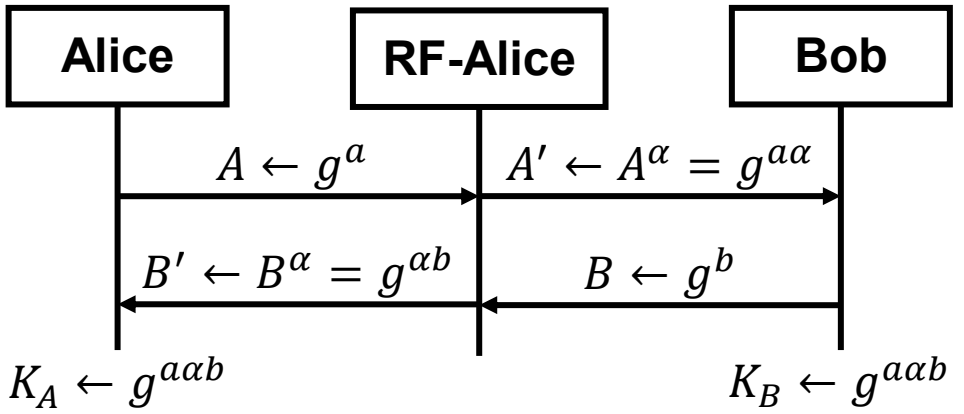
Subversion-resilient AKE – Example 1 :

2015 – Secure Transmission with Reverse Firewalls – Protecting Alice

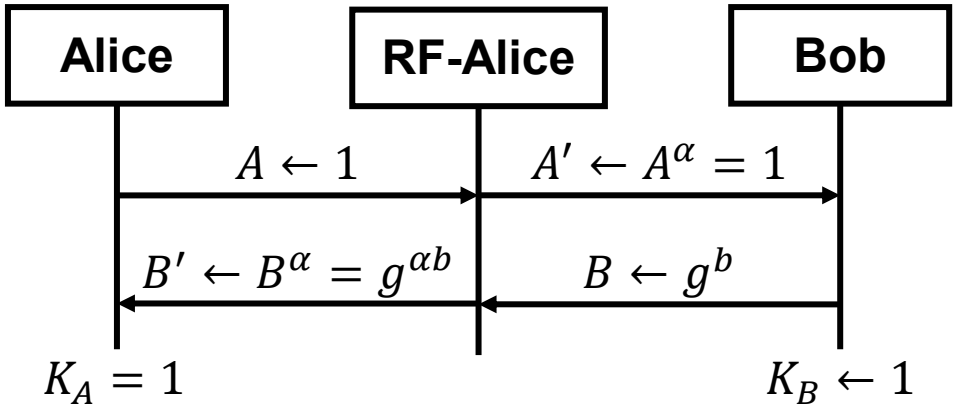
The idea

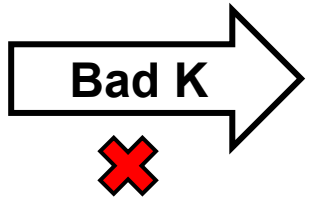


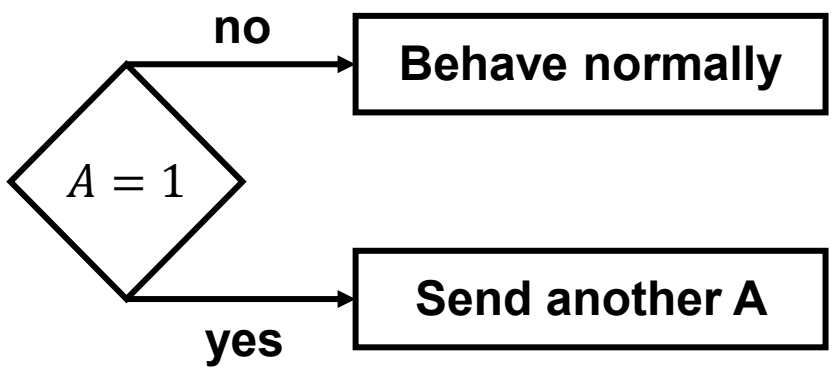




Another issue



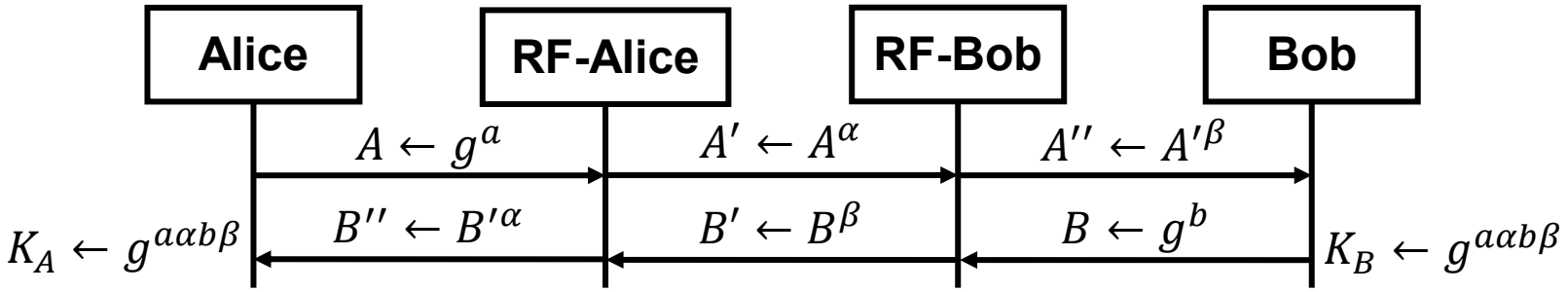




Subversion-resilient AKE – Example 1 :

2015 – Secure Transmission with Reverse Firewalls – Protecting Bob

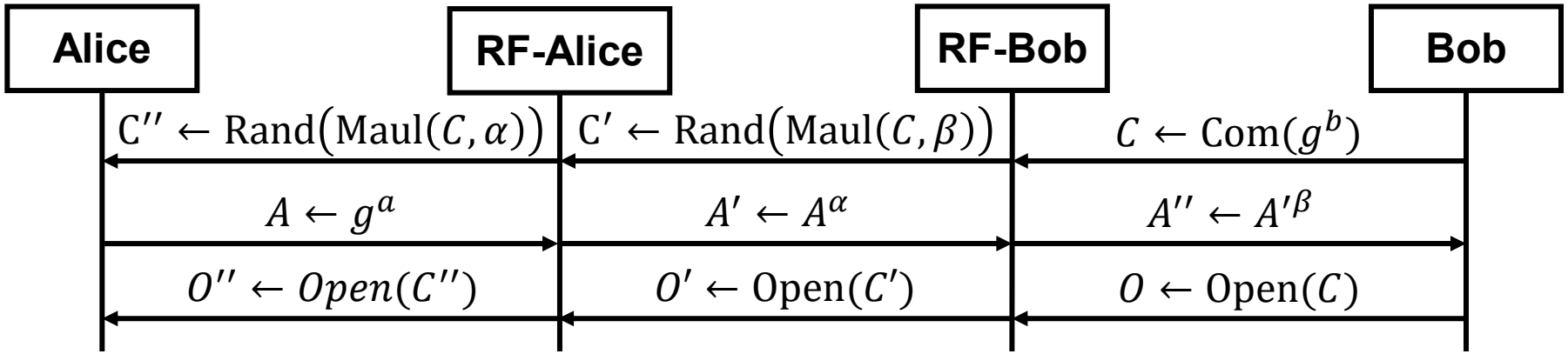
A first idea that does not work : RF-Bob = RF-Alice



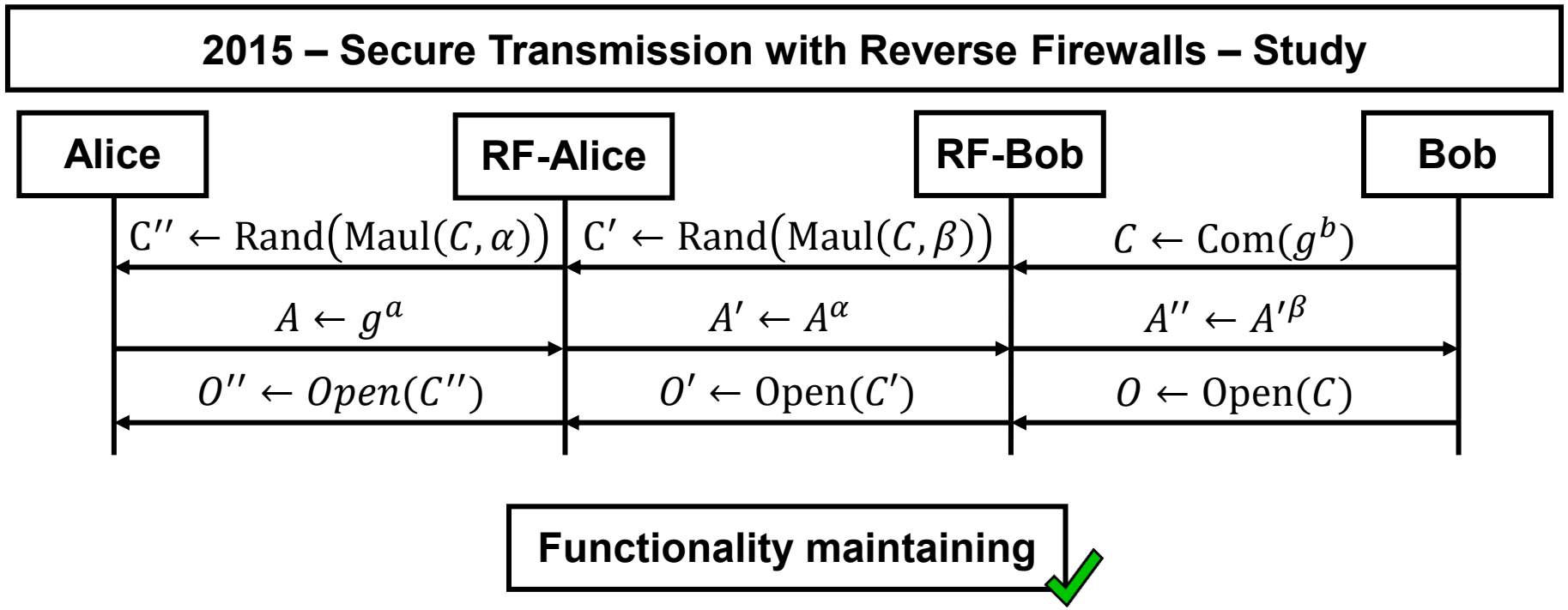
Rejection sampling on b :
$H(K) = 0 \ldots 0 \,||\, rand$

Ideas (but bad) : randomizing K, Bob starting the protocol

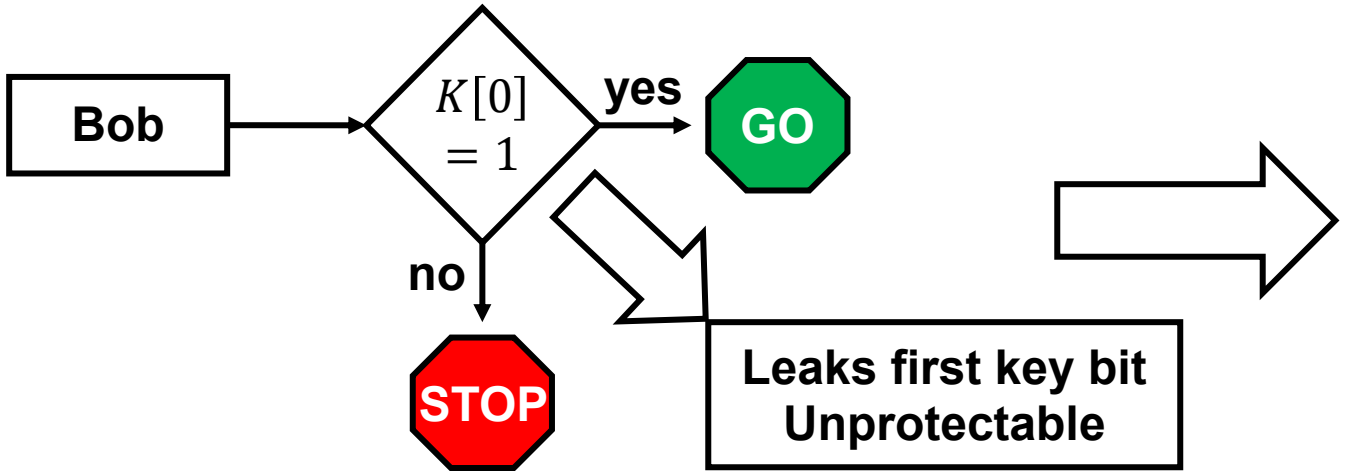
The solution



Subversion-resilient AKE – Example 1 :



Functionality maintaining ✓



Functionality maintaining subversions

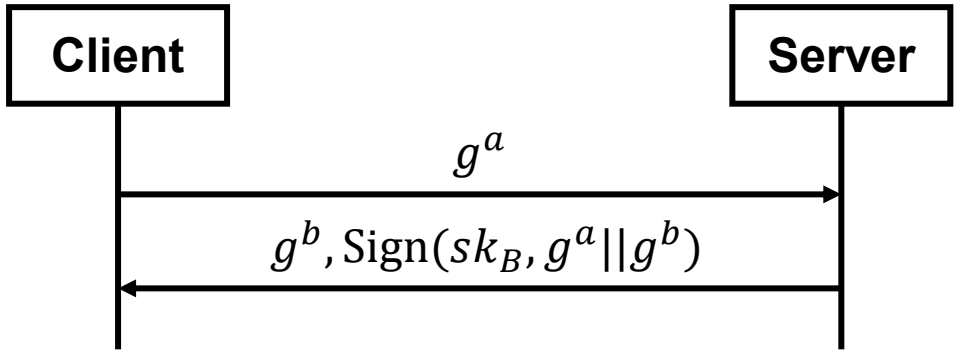
Security preservation ✓

Exfiltration-resilience ✓

Subversion-resilient AKE – Example 2 :

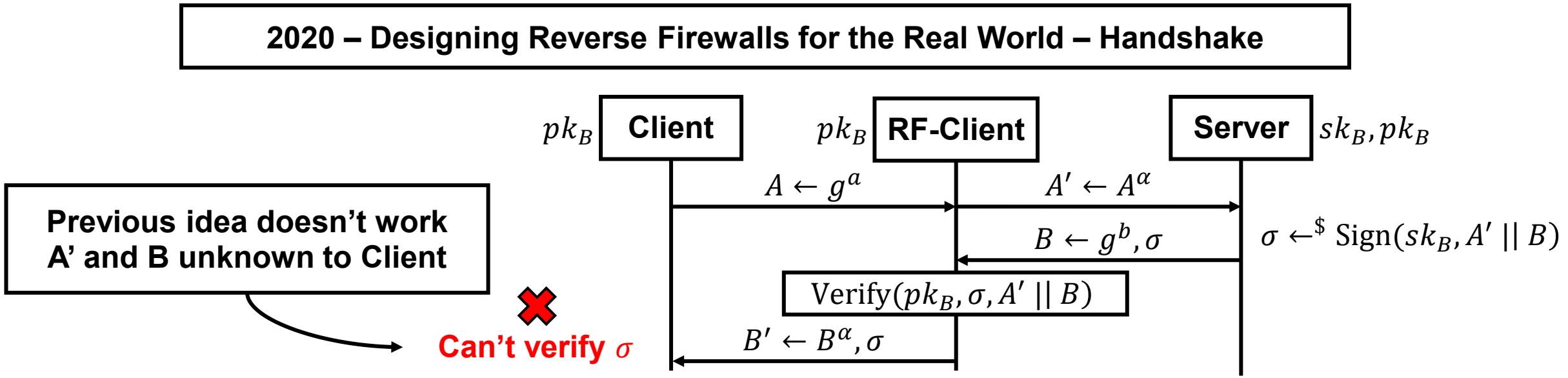
2020 – Designing Reverse Firewalls for the Real World

To protect : server-authenticated DH (TLS)

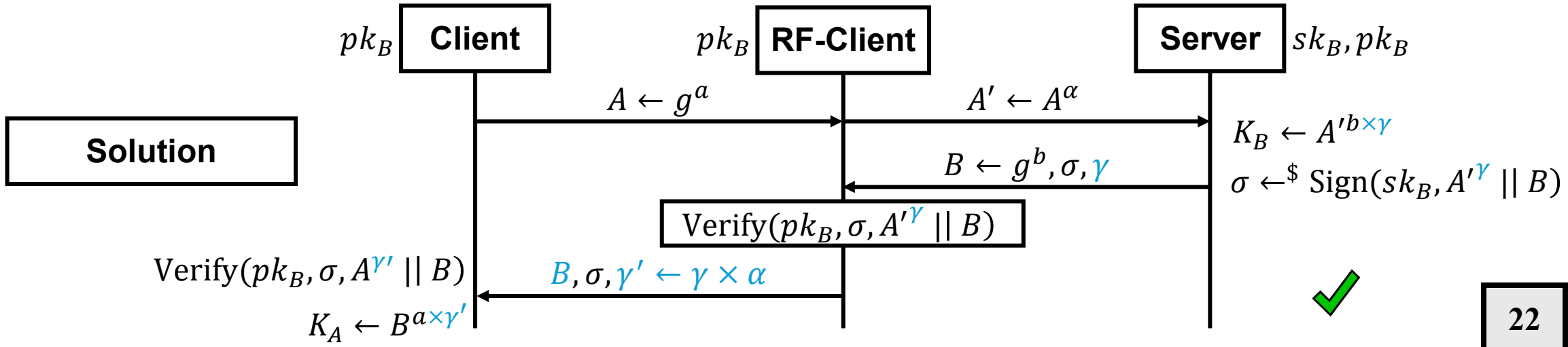


- ✚ No commitment scheme
- ✚ No pairings
- ✚ 1 round trip
- ✚ Reinforced models
- ➖ Only client-side (but better for NSA)

Subversion-resilient AKE – Example 2 :



Ideas : randomizing signature content (bad), RF reveal exponent to client (maybe good)



Subversion-resilient AKE – Example 2 :

2020 – Designing Reverse Firewalls for the Real World – Message exchange

Previous paper

« Functionality maintaining subversions » with unique encryption, cheat :

$\mathrm{Dec}(\mathrm{Enc}(m,k),k) \neq m$: not functionality maintaining

Excludes

- Encrypt messages with attacker key
- Rejection sampling on m (add spaces)
- Input trigger

Solution : double-layered encryption ✓

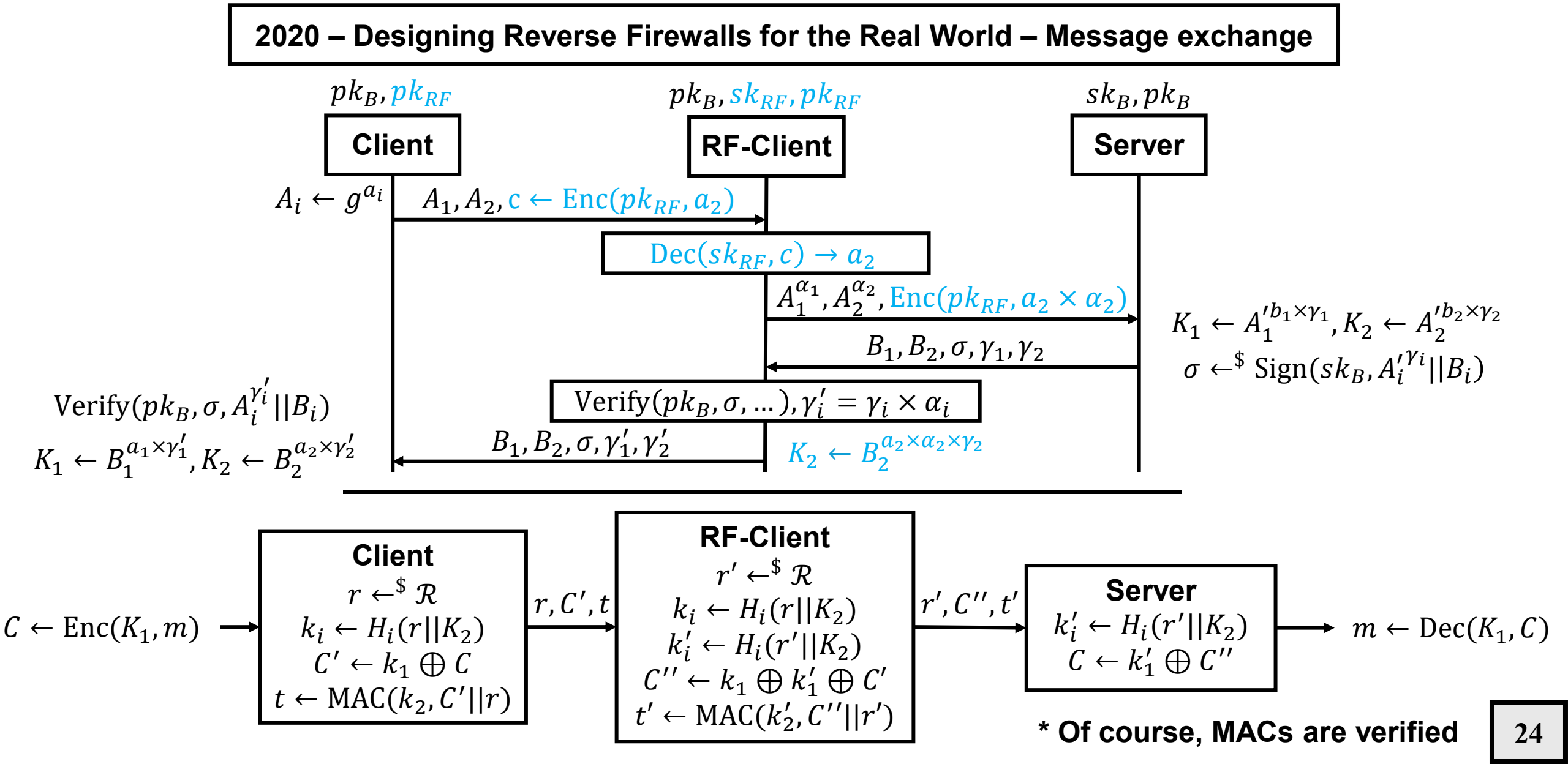
Client & Server → K, K'

RFs → K'

RFs can rerandomize SymEnc with K'

- Client OK, RF BAD : layer 1
- Client BAD, RF OK : layer 2

Subversion-resilient AKE – Example 2 :



* Of course, MACs are verified

Subversion-resilient AKE – Example 2 :

2020 – Designing Reverse Firewalls for the Real World – Exfiltration

Before ✗

Client : $b \leftarrow^{\$} \{0,1\}$
$b = 0$ **: implem**
$b = 1$ **: spec**
⇔ **RF** ⇔ $\mathcal{A}$ **must find** b

But spec ideal : real world => reference implementation

After ✓

Client : $b \leftarrow^{\$} \{0,1\}$
$b = 0$ **: implem 1**
$b = 1$ **: implem 2**
⇔ **RF** ⇔ $\mathcal{A}$ **must find** b

Trivial attacks : P1 does nothing, P2 honest

… but « functionality maintaining » is cheating

=> Transcript equivalence

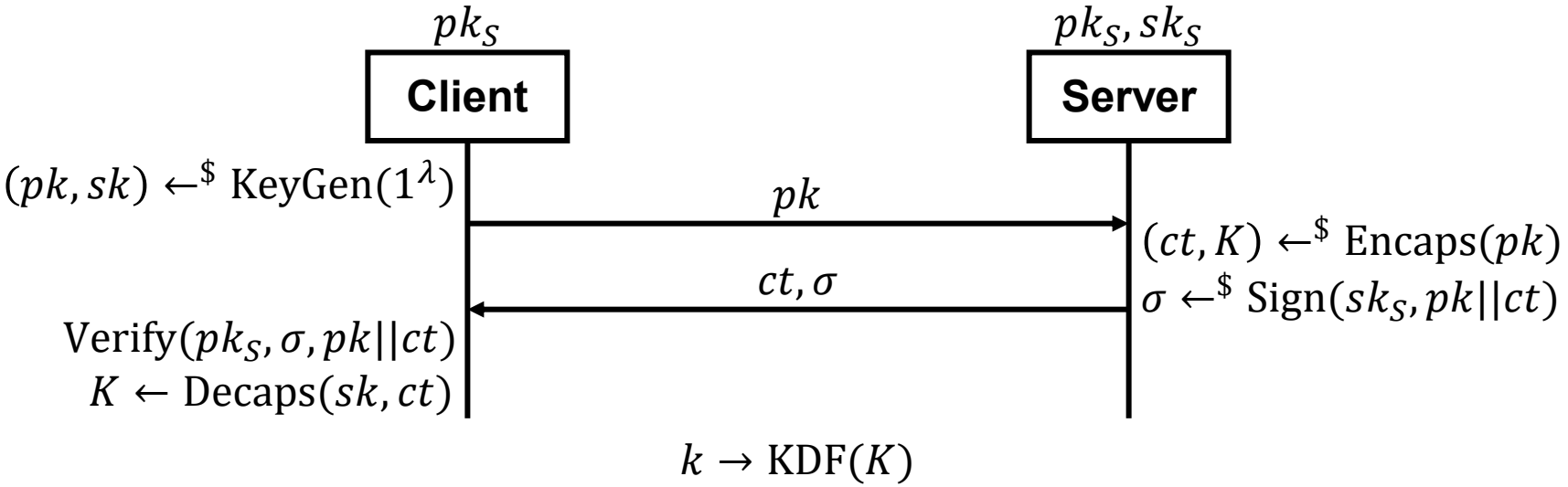
PQ SR-AKE – Introduction :

Goal : same as before but for PQ TLS

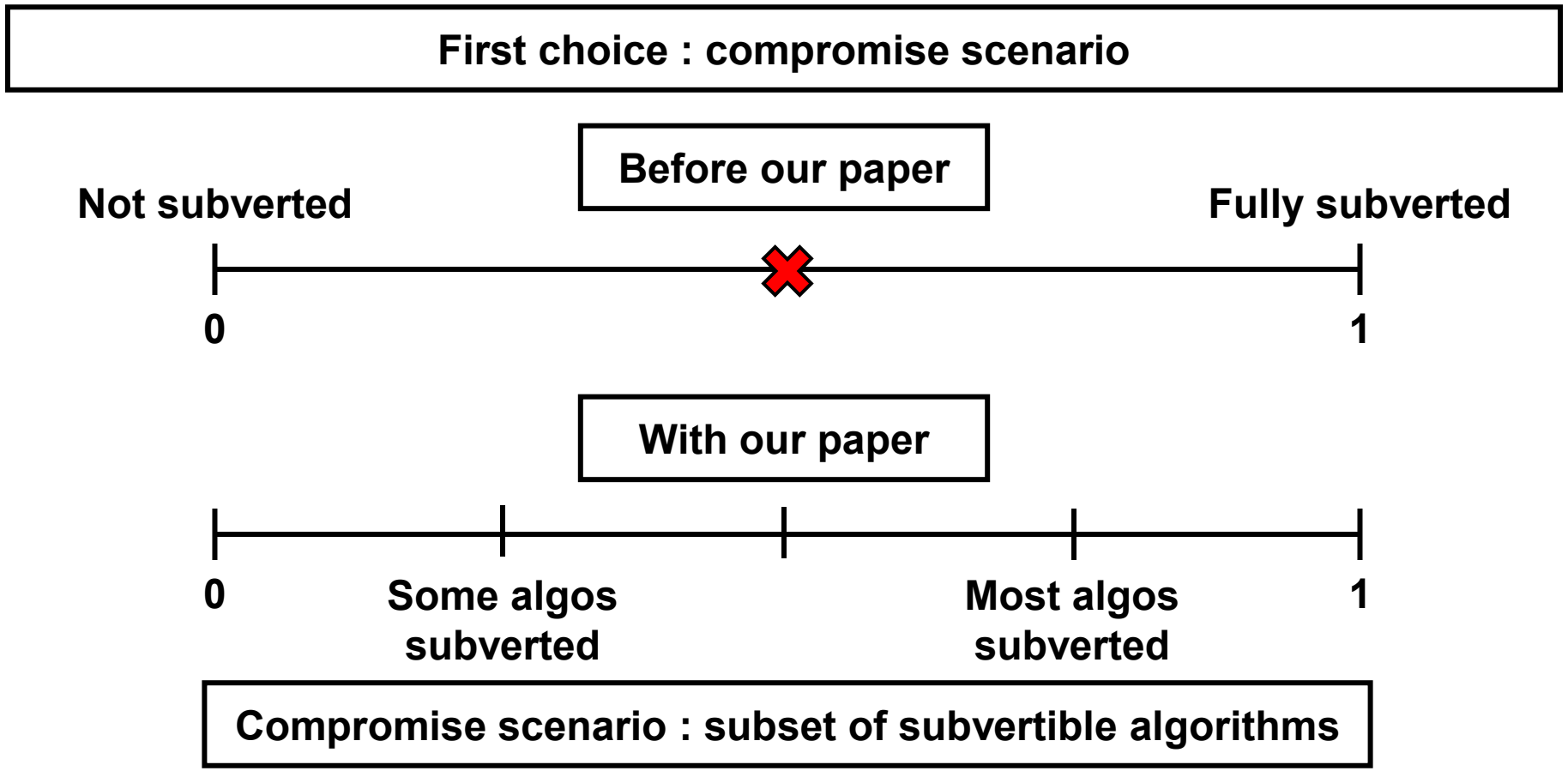
Main issue : no PQ DH !

KEM

Client-authenticated KEM key-exchange



PQ SR-AKE – Design choices :



- More efficient protocols (no useless protections)
- Spectrum of protocols (for each trust scenario)

PQ SR-AKE – Design choices :

Second choice : setup-free RFs

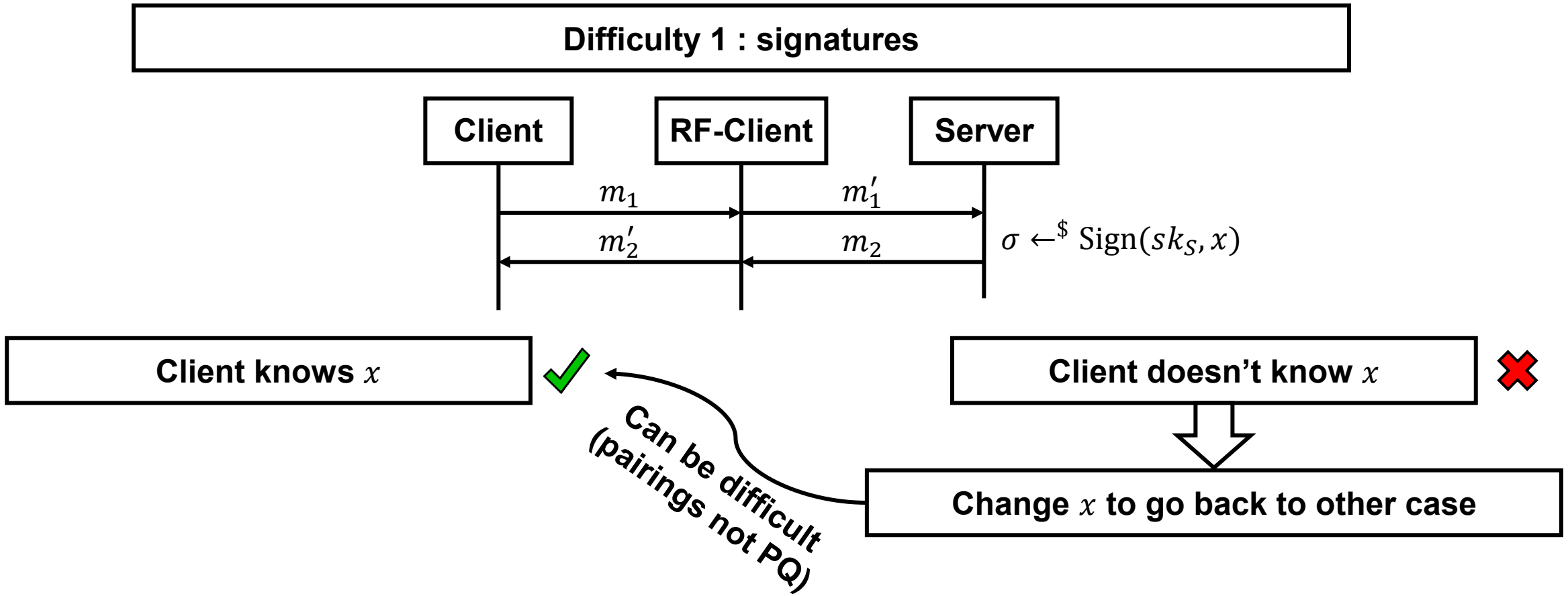
2015 – Secure Transmission with Reverse Firewalls	**RF has no (pk_{RF}, sk_{RF})**
2020 – Designing Reverse Firewalls for the Real World	**RF has (pk_{RF}, sk_{RF})**

Setup free : RF with no (pk_{RF}, sk_{RF})

- **Easier to deploy**
- **Naturally stackable RFs**

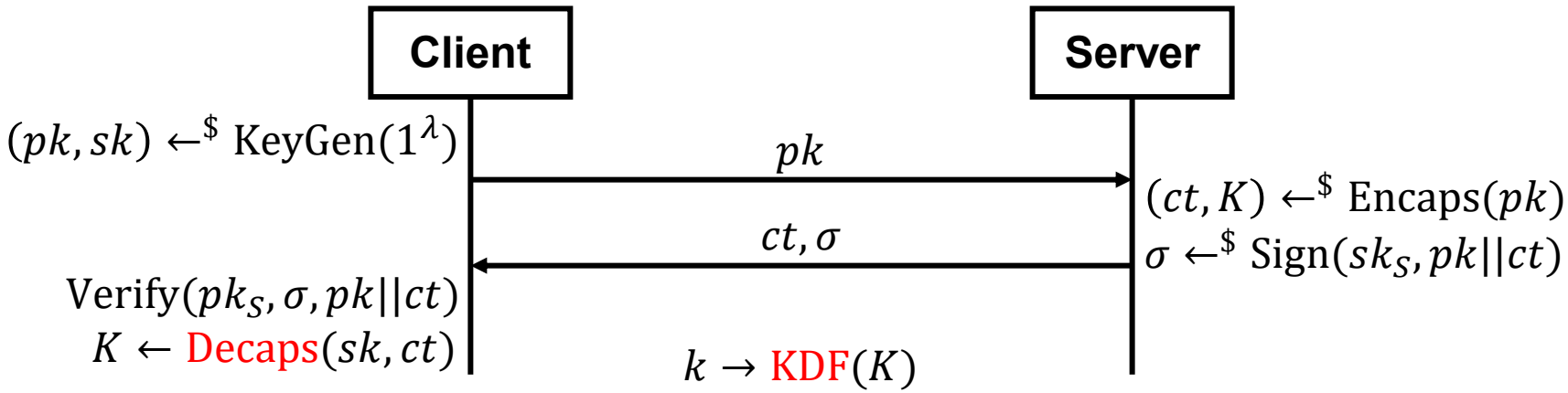
Goal : only setup-free RFs !

PQ SR-AKE – Design choices :



PQ SR-AKE – Design choices :

Difficulty 2 : Decaps and KDF



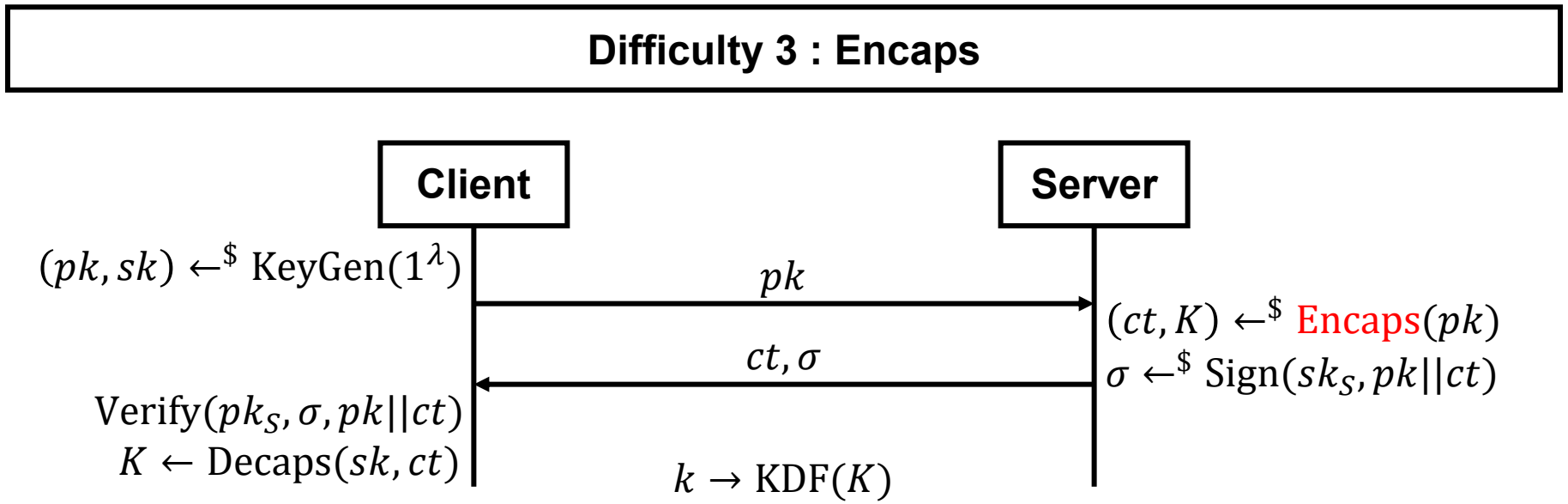
Could choose k independent of handshake

Existing solution : double layered encryption !

Adaptation to PQ (possible) ✓

Setup free version ? ✗

PQ SR-AKE – Design choices :



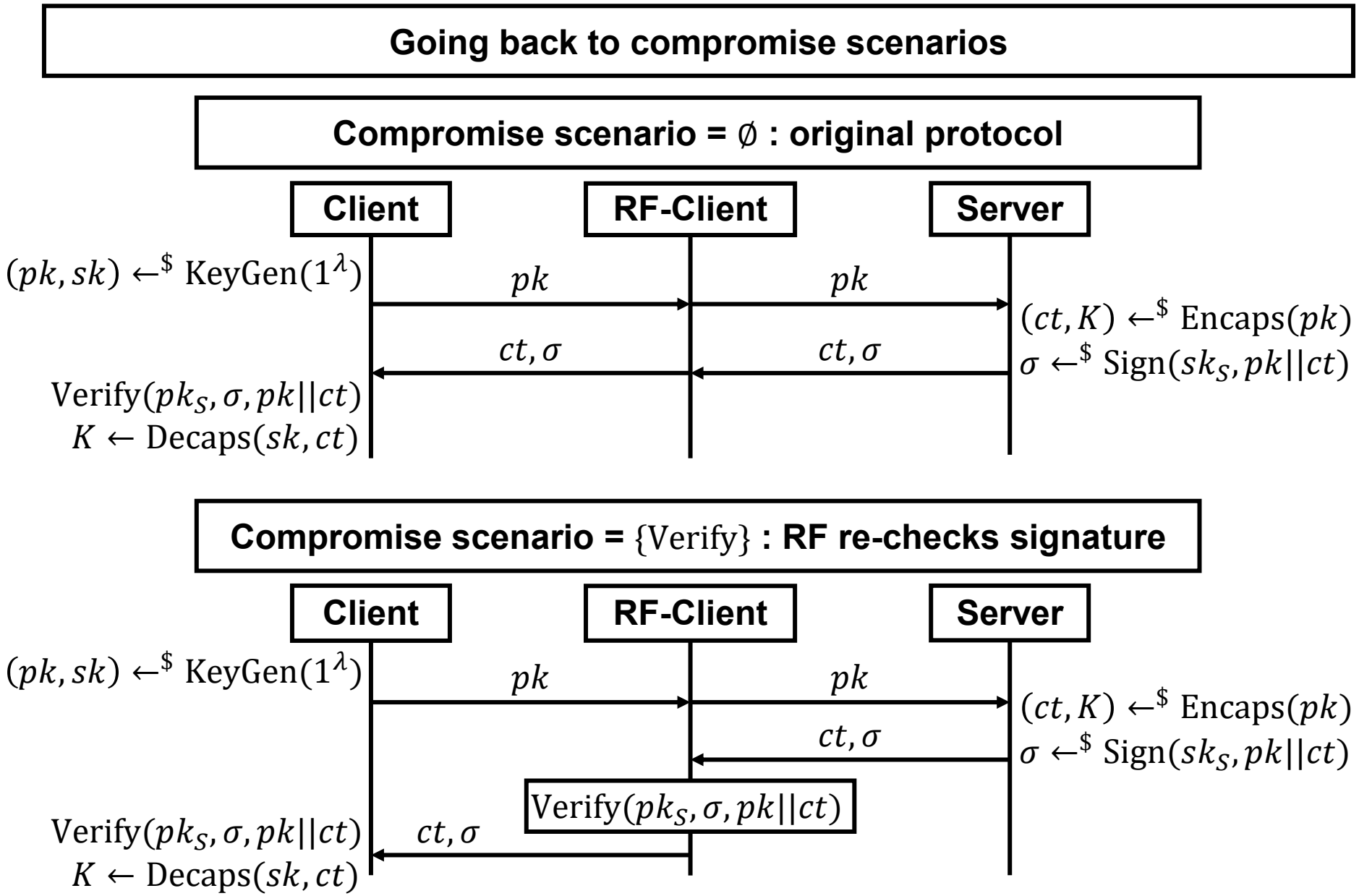
Easy : Rejection sampling

Hard : Unauthenticated client

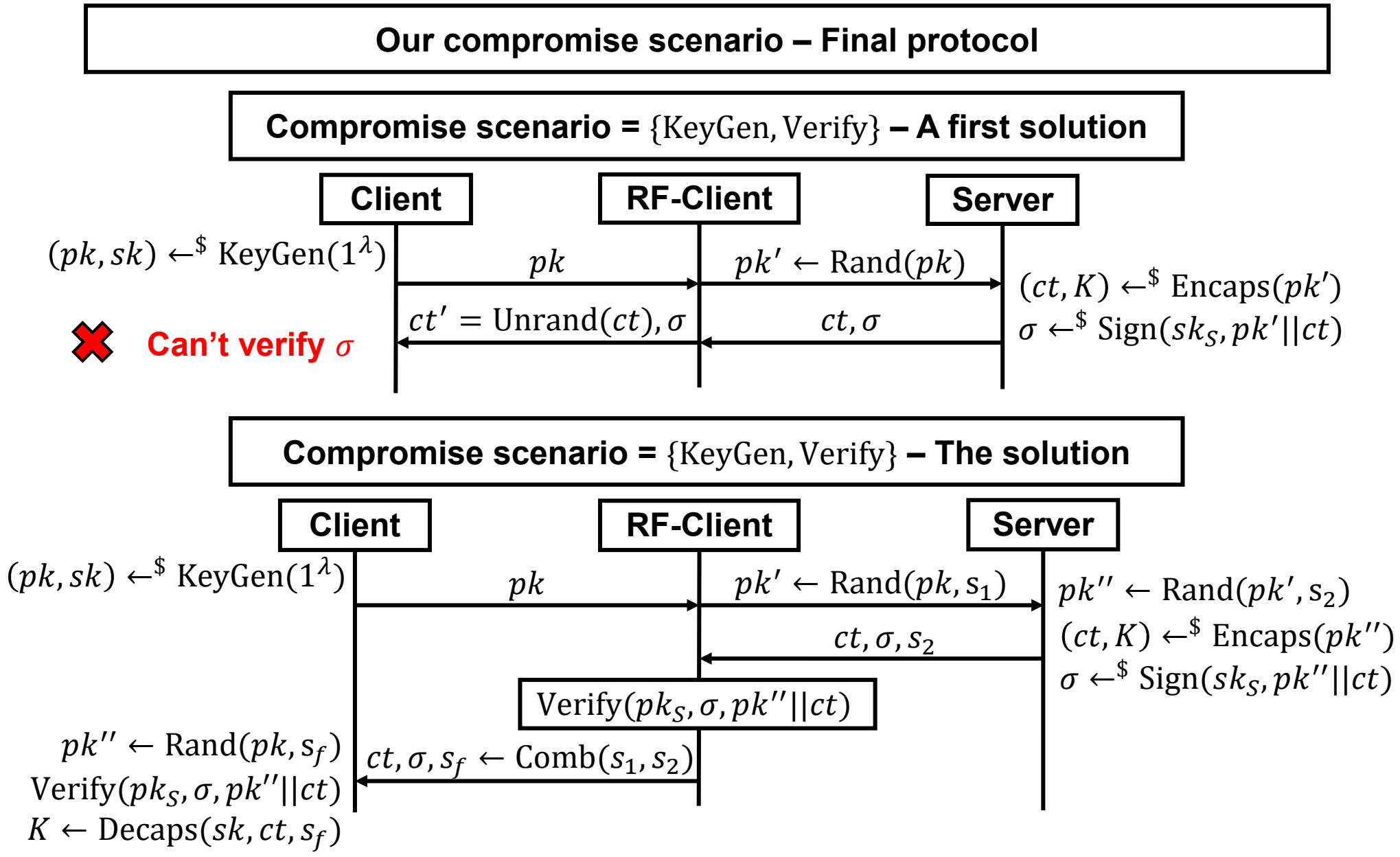
Issue of (some) KEMs : server can unilaterally choose K !

Solution : commitment like 2015, does not work for all KEMs + inefficient !

PQ SR-AKE – The protocol :



PQ SR-AKE – The protocol :



PQ SR-AKE – Rerandomizable KEM :

Syntax

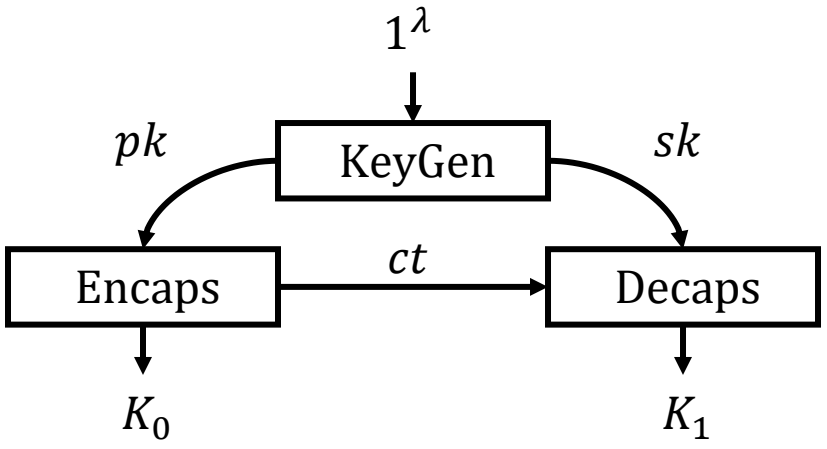
"Normal" KEM

$(pk, sk) \leftarrow^{\$} \text{KeyGen}(1^\lambda)$
$(ct, K) \leftarrow^{\$} \text{Encaps}(pk)$
$K \leftarrow \text{Decaps}(sk, ct)$

Rerandomizable KEM

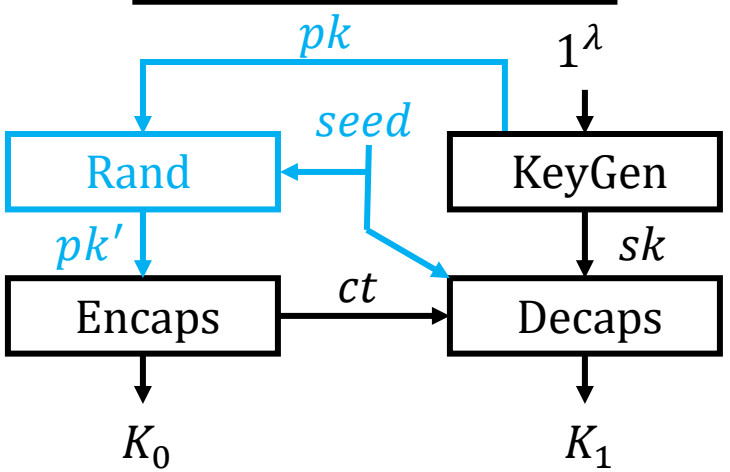
$(pk, sk) \leftarrow^{\$} \text{KeyGen}(1^\lambda)$
$(ct, K) \leftarrow^{\$} \text{Encaps}(pk)$
$K \leftarrow \text{Decaps}(sk, ct, seed)$
$pk' \leftarrow \text{Rand}(pk, seed)$
$seed' \leftarrow \text{Comb}(seed_1, seed_2)$

Correctness



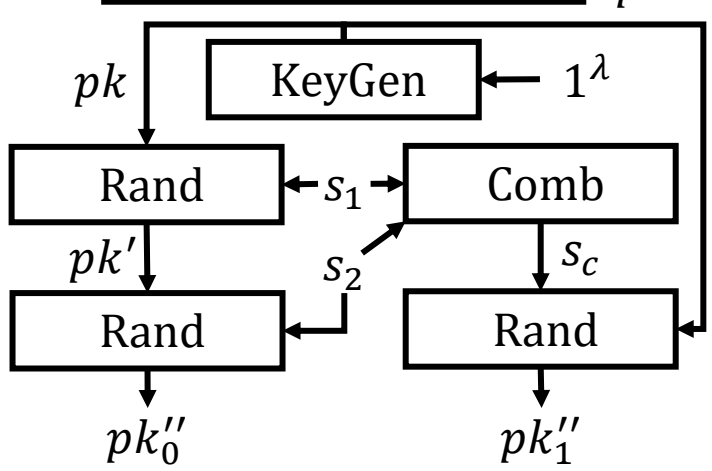
$K_0 = K_1$

Usual correctness



$K_0 = K_1$

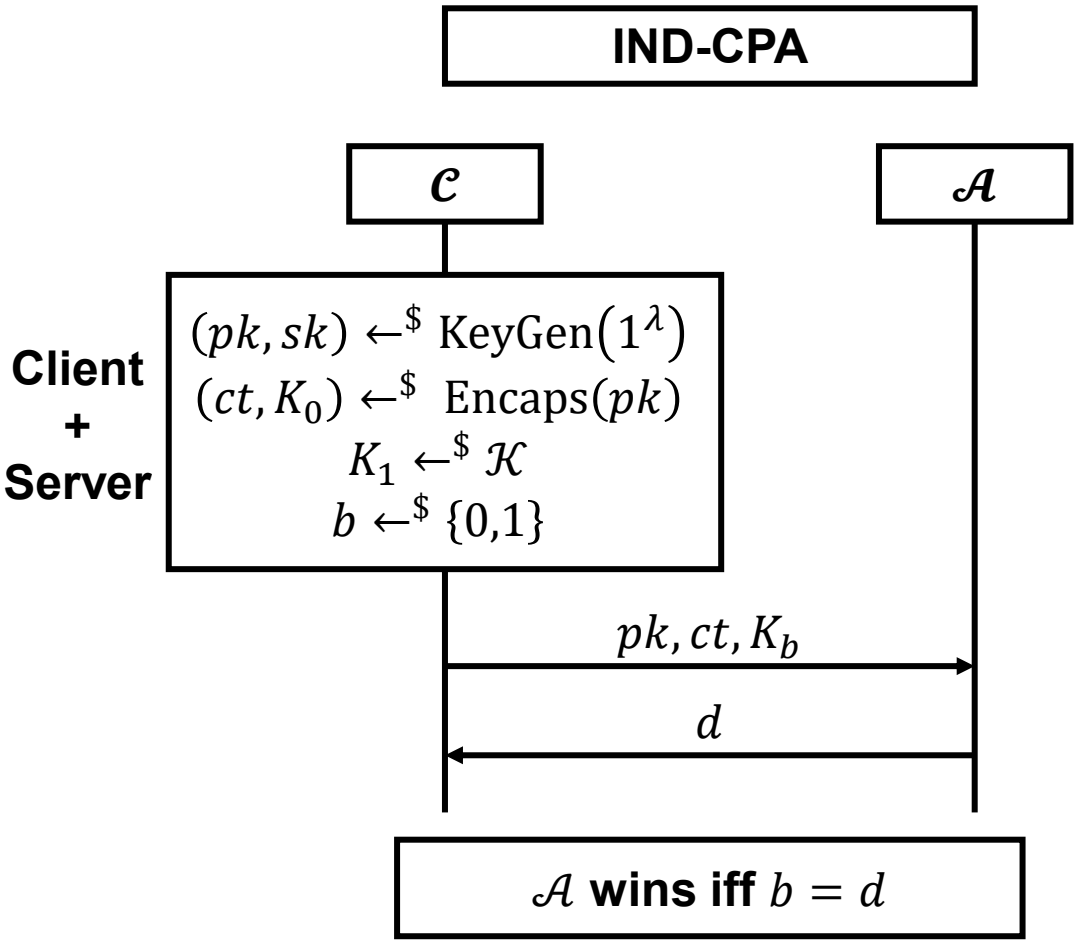
Comb correctness

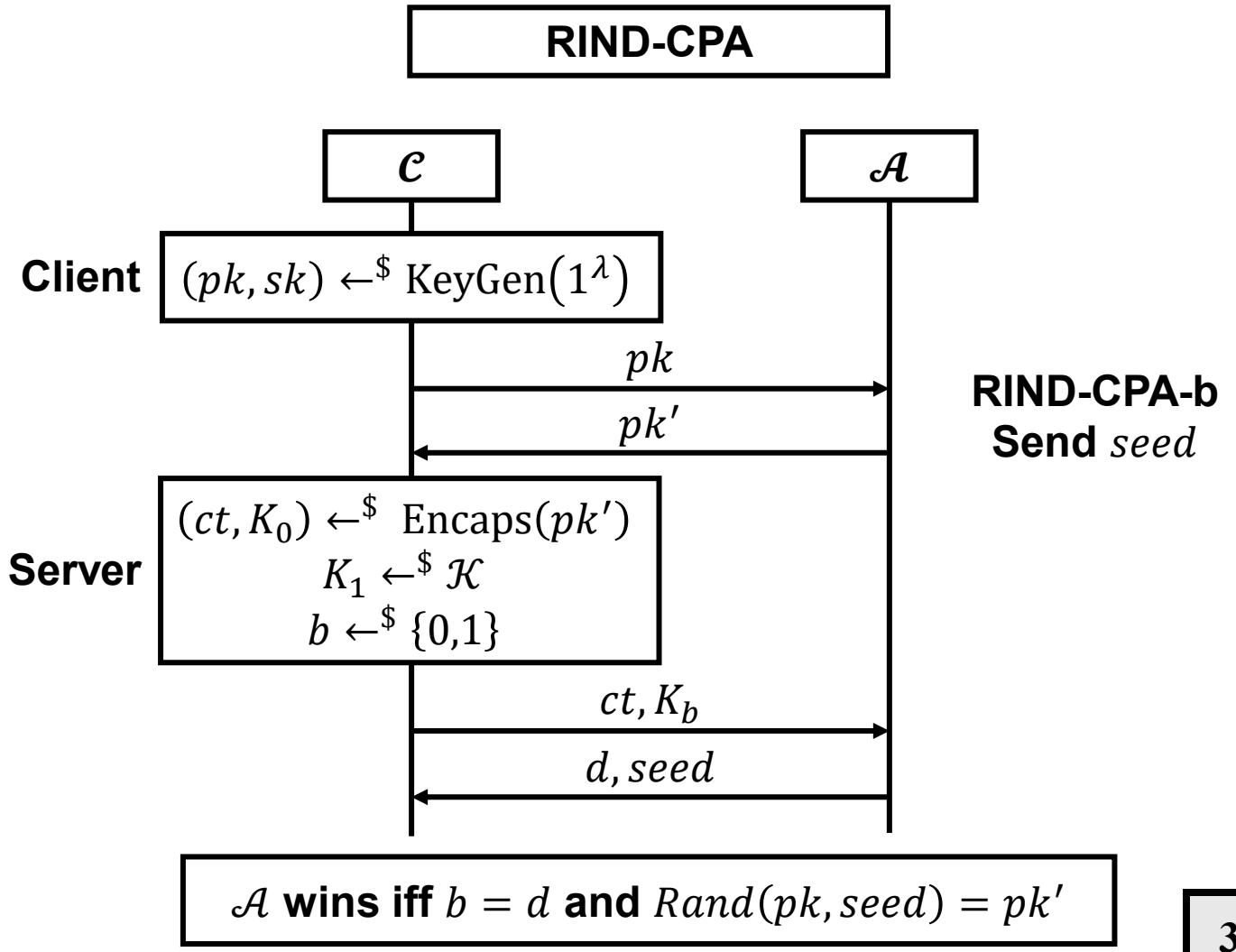


$pk_0'' = pk_1''$

PQ SR-AKE – Rerandomizable KEM :

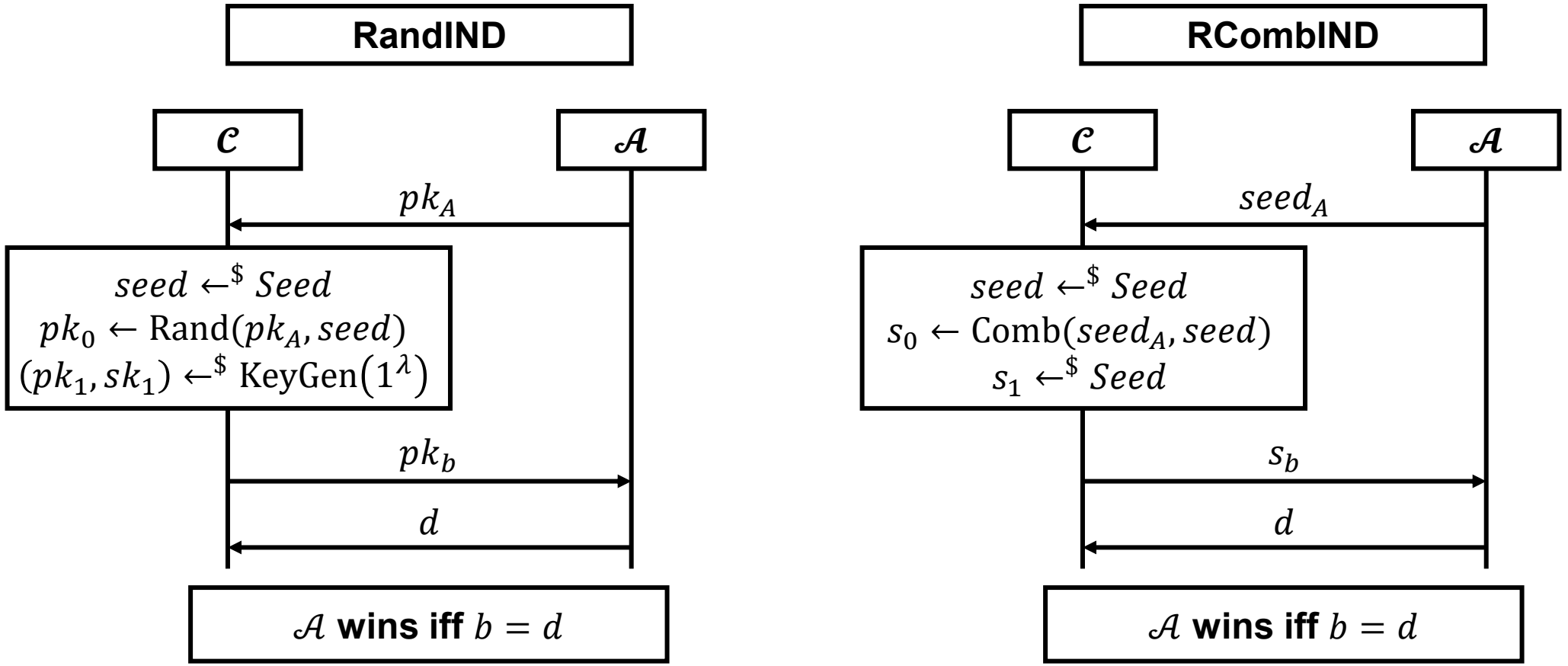
Security definitions : IND-CPA too weak, IND-CCA too strong => RIND-CPA !





PQ SR-AKE – Rerandomizable KEM :

Security definitions : for exfiltration-resilience



PQ SR-AKE – Rerandomizable KEM :

Instanciations

DH-based

Global parameters : g

$\mathrm{KeyGen}(1^\lambda) \rightarrow (x, g^x)$
$\mathrm{Encaps}(pk) \rightarrow (g^r, pk^r)$

$\mathrm{Rand}(pk, seed) \rightarrow pk^{seed}$
$\mathrm{Comb}(s_0, s_1) \rightarrow s_0 \times s_1$

$\mathrm{Decaps}(sk, ct, seed) \rightarrow ct^{sk \times seed}$

Lattice-based

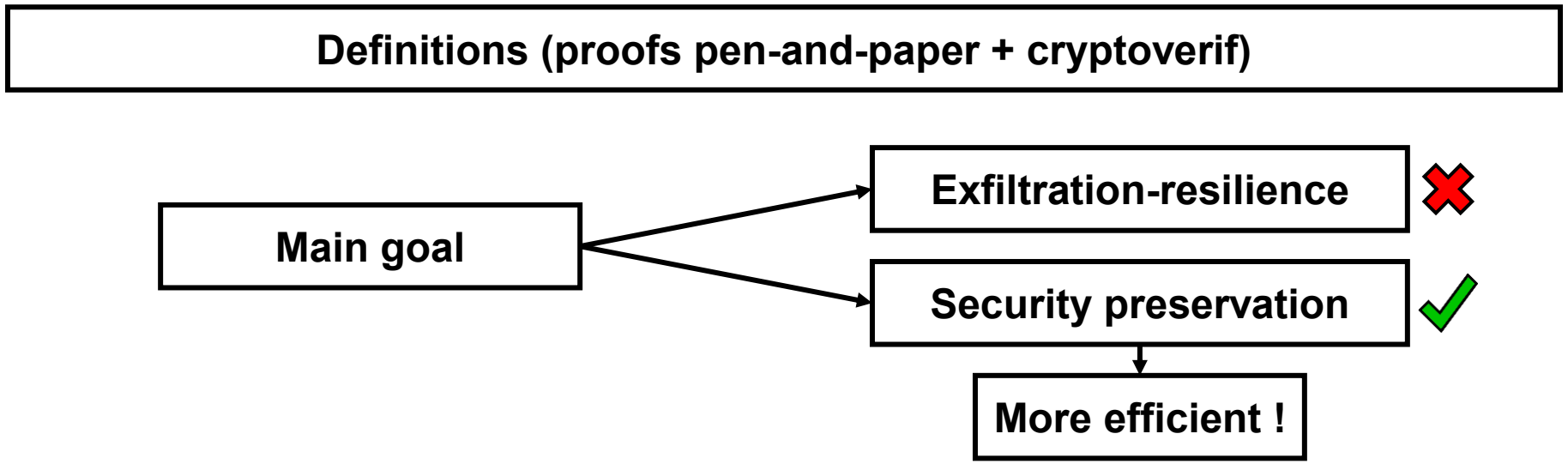
Global parameters : A

$\mathrm{KeyGen}(1^\lambda) \rightarrow \big((s, e), As + e\big)$
$\mathrm{Encaps}(pk) \rightarrow (ct, K)$

$\mathrm{Rand}(pk, (s', e')) \rightarrow pk + As' + e'$
$\mathrm{Comb}\big((s_0, e_0), (s_1, e_1)\big) \rightarrow (s_0 + s_1, e_0 + e_1)$

$\mathrm{Decaps}(sk, ct, seed) \rightarrow \mathrm{UsualDecaps}(sk + seed, ct)$

PQ SR-AKE – Security definitions :



AKE with RFs : usual security with malicious RF

Authenticating / Securing : security preservation

Exfiltration-resilience

Obliviousness

PQ SR-AKE – Security definitions :

AKE with RFs

Usual AKE : NewInstance, Send, Reveal, Corrupt, Test

Additionnal powers

- **NewRF : RF party creation**
- **SetupRF : link RFs to I / R**

Setup-free RFs : we can ignore them !

The proof :

Step 1 – authentication	Step 2 – key secrecy
EUF-CMA signature Not same SID (initiator)	Not same SID (responder) RIND-CPA KEM

PQ SR-AKE – Security definitions :

Security preservation (authenticating and securing)

Usual AKE : NewInstance, Send, Reveal, Corrupt, Test

Additionnal power : SetupRF, SetCode subverted parties

+ Attacker can’t view messages between subverted parties and their RFs

The proof :

Step 1 – authenticating	Step 2 – securing
RF re-checks signature Rerand. => not same SID I	Identical to previously (need to replace pk’ honest)

PQ SR-AKE – Security definitions :

Exfiltration-resilience / Obliviousness

Oracles : NewInstance, Send, Reveal, SetupRF

Exfiltration-resilience : SetCode LoR (⇔ transcripts)

Obliviousness : modified NewInstance LoR

+ Attacker can’t view messages between subverted parties and their RFs

The proof (both cases) :

Replace pk’ by honest one
Needs to delete sk and pk use on I => Replace K on I by partner R

Advantage Obliviousness < Advantage Exf-Res in reality

Conclusion :

Expand compromise scenario

- **Example : adapt double-layer encryption (setup-free RF)**

Get close to real TLS

- **Nonces**
- **Legacy fields : version, session IDs, …**
- **Already considered (not fully) in 2020 paper**

Develop solutions to trust algorithms

- **Watchdogs, implementation verification, …**

Thank you for your attention !