Seems Legit

Automated Analysis of Subtle Attacks on Protocols that Use Signatures

Dennis Jackson

UNIVERSITY OF OXFORD

Katriel Cohn-Gordon

Cas Cremers

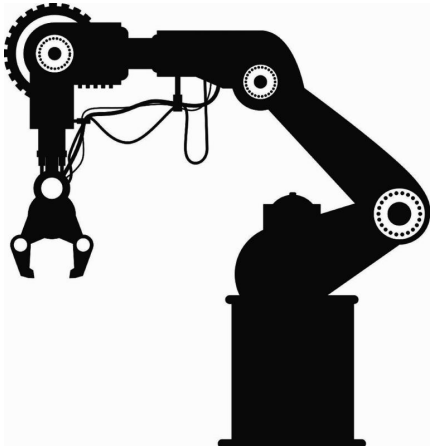
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Ralf Sasse

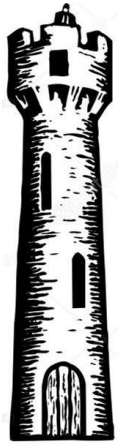
ETH zürich

This Talk

Subtle
Signature
Behaviours



Automated
Protocol
Analysis



Real
World
Attacks

Subtle Behaviours

Definition: Signature Scheme

Three algorithms:

```
Gen()                  -> (sk,pk)

Sign(sk,msg)           -> sig

Verify(sig,msg,pk)  -> {true,false}
```

Correct if for all messages **`m`** and any **`(sk,pk) <- Gen()`**

- **`Verify(Sign(m,sk),m,pk) = true`** (almost always)

Definition: Unforgeability

Existential unforgeability under an adaptive chosen message attack

1. The referee generates a keypair and outputs the public key
2. The adversary may (adaptively) ask the referee for a signature on a message of the adversary's choice.
3. The adversary wins if they can produce a message and signature pair which passes **`Verify`**, but the adversary never submitted the message in step 2.

Introduced[1] in **1988**, widely accepted as the standard definition.

[1] Goldwasser, S., Micali, S., & Rivest, R. L. (1988)

Substitutions

Key Substitution[1] (1999)

Given `sig,pk,msg` with

`verify(sig,msg,pk) = true`

Calculate `sk',pk'` such that

`verify(sig,msg,pk') = true`

Message-Key Substitution[2] (2000)

Given `sig,pk,msg,msg'` with

`verify(sig,msg,pk) = true`

Calculate `sk',pk'` such that

`verify(sig,msg',pk') = true`

[1] Blake-Wilson, S., & Menezes, A. (1999). [2] Baek, J., & Kim, K. (2000)

Toy Protocol I

Start: **i** and **r** have a shared secret **k** (exchanged out of band)
Goal: **r** receives and authenticates **i**'s public key and message **msg**

1. **i** creates **sig := sign$_i$(msg)** and **tag := mac$_k$(s)**
2. **i** sends: **msg, pk$_i$, sig, tag**
3. **r** checks: **mac$_k$(sig) == tag**
4. **r** checks: **verify(sig, msg, pk$_i$) == true**

Attack on Toy Protocol I

Start: **i** and **r** have a shared secret **k** (exchanged out of band)
Goal: **r** receives and authenticates **i**'s public key and message **msg**

1. **i** creates $\mathbf{sig := sign_i(msg)}$ and $\mathbf{tag := mac_k(s)}$
2. **i** sends: $\mathbf{msg, pk_i, sig, tag}$

- Attacker calculates $\mathbf{pk_m := MKS(sig, msg_m)}$
- Attacker forwards: $\mathbf{msg_m, pk_m, sig, tag}$

1. **r** checks: $\mathbf{mac_k(sig) == tag}$
2. **r** checks: $\mathbf{verify(sig, msg_m, pk_m) == true}$

r is confused about the identity of the initiator and accepted a forged message!

Colliding Signatures[1] (2002)

Given **`msg,msg' and msg != msg'`**

Calculate **`sk,pk,sig`** such that

`verify(sig,msg ,pk) = true`

`verify(sig,msg',pk) = true`

Calculate **`sk,pk,sig`** such that for random **`msg`** non-negligible probability that

`verify(sig,msg,pk) = true`

[1] Stern, J., Pointcheval, D., Malone-Lee, J., & Smart, N. P. (2002).

Toy Protocol II

Start: **i** knows **r**'s public key $\mathbf{pk_r}$
Goal: **i** and **r** share a secret key **k** and **r** knows **i**'s public key $\mathbf{pk_i}$

1. **i** creates $\mathbf{ct := aenc_r(k)}$ and $\mathbf{sig := sign_i(k)}$
2. **i** sends $\mathbf{pk_i, ct, sig}$
3. **r** decrypts the ciphertext **ct** and learns **k**
4. **r** checks: $\mathbf{verify(sig, k, pk_i) == true}$

Attack on Toy Protocol II

Start: **i** knows **r**'s public key $\mathbf{pk_r}$
Goal: **i** and **r** share a secret key **k** and **r** knows **i**'s public key $\mathbf{pk_i}$

1. **i** creates $\mathbf{ct := aenc_r(k)}$ and $\mathbf{sig := sign_i(k)}$
2. **i** sends $\mathbf{pk_i}$, **ct**, **sig**
   - Attacker creates $\mathbf{pk_m}$, $\mathbf{sig_m}$:= **collide()**
   - Attacker forwards $\mathbf{pk_m}$, **ct**, $\mathbf{sig_m}$
1. **r** decrypts the ciphertext **ct** and learns **k**
2. **r** checks: $\mathbf{verify(sig_m, k, pk_m)}$ == **true**

This is an **unknown key share** attack. **r** is confused about the identity of **i**, but **k** is not known to the adversary

Behaviours

Key Substitution

Given a signature, produce a new public key which also verifies that signature

Message-Key Substitution

Given a signature, produce a new public key which also verifies that signature under ***a different message.***

Colliding Signatures

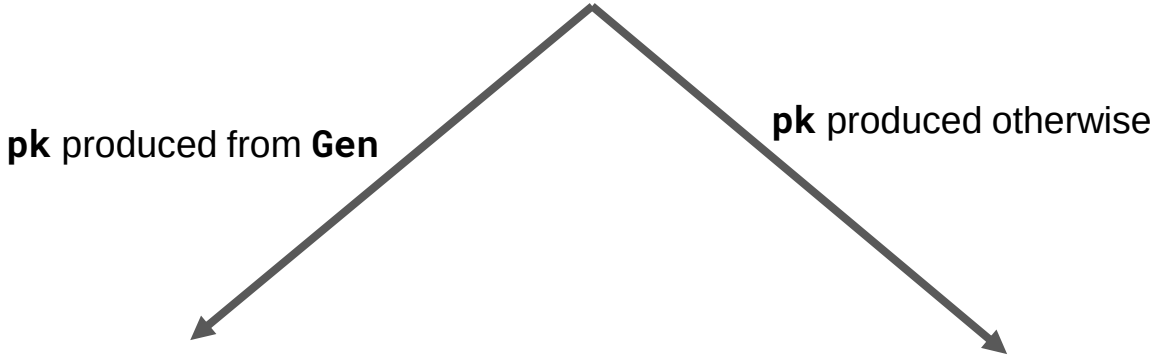
Compute a public key and signature which verifies for more than one message

(Strong) Colliding Signatures

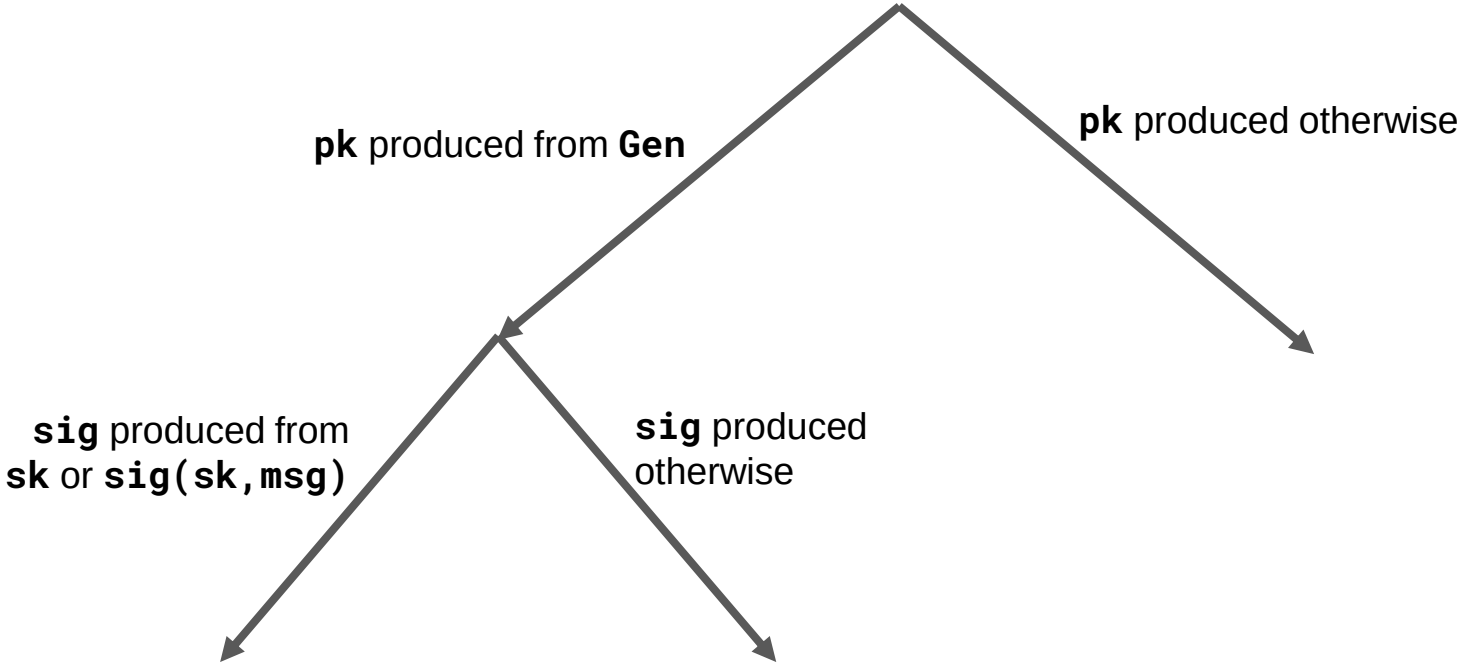
Compute a public key and signature which verifies for ***any message***

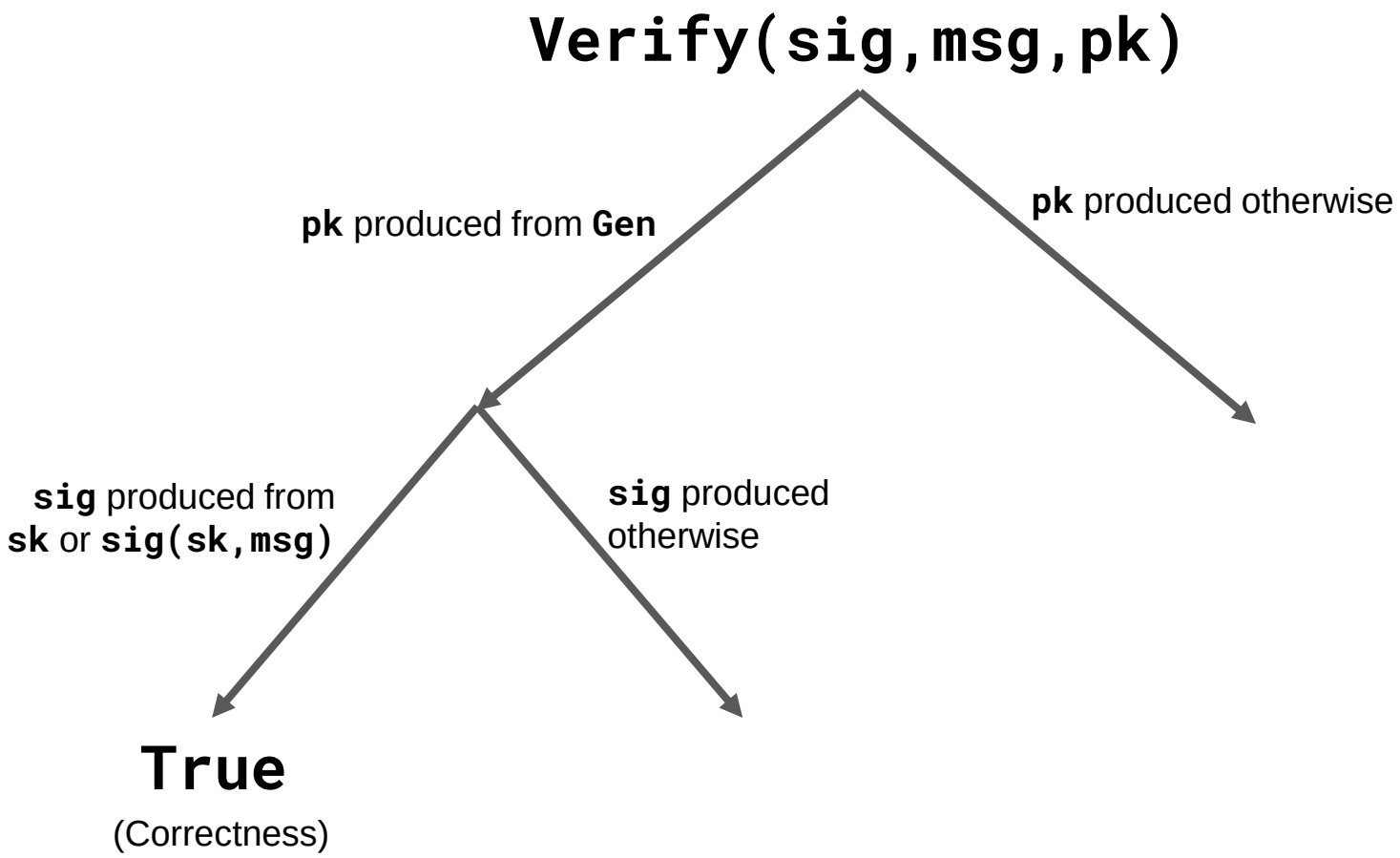
Verify(sig,msg,pk)

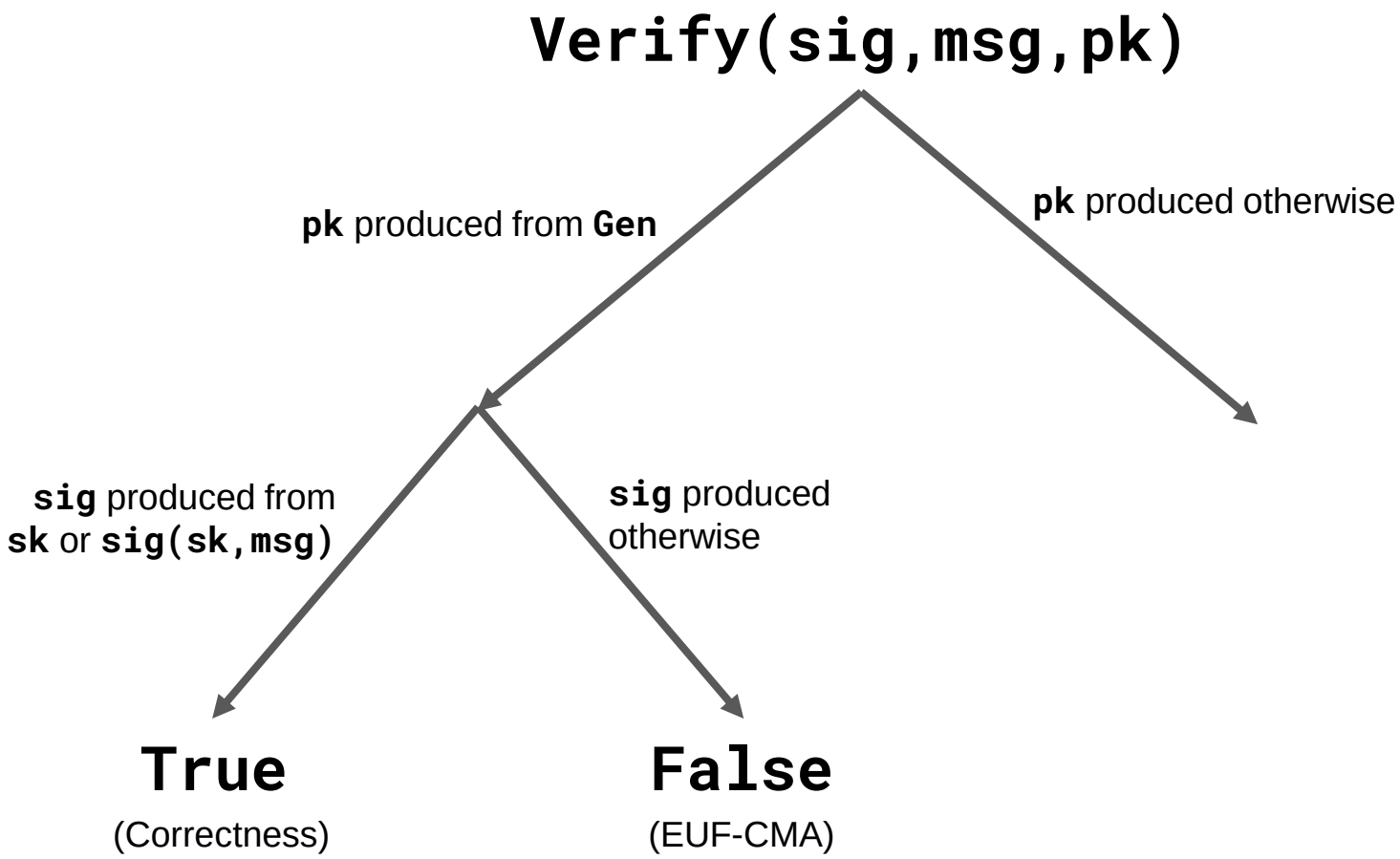
`Verify(sig,msg,pk)`

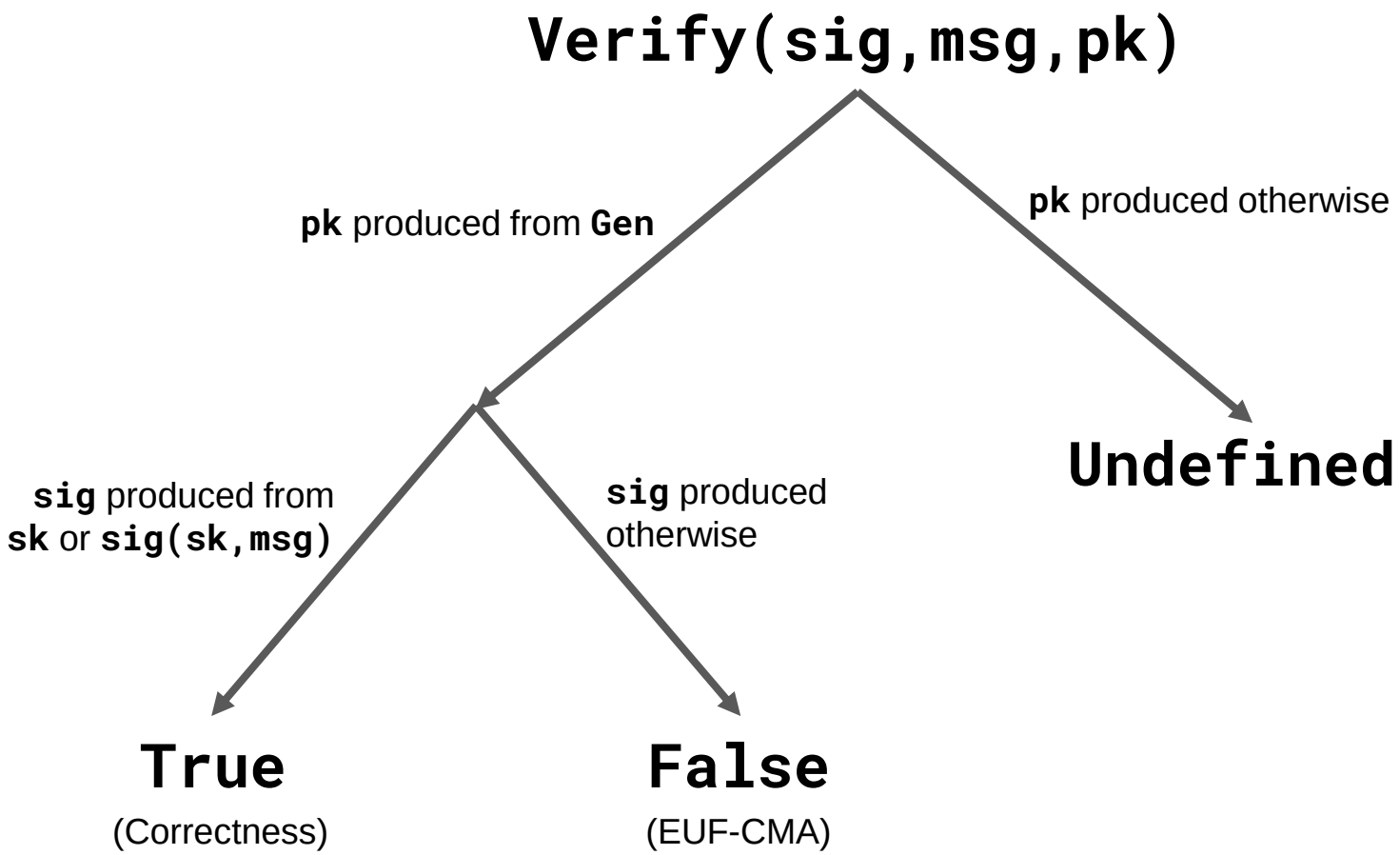


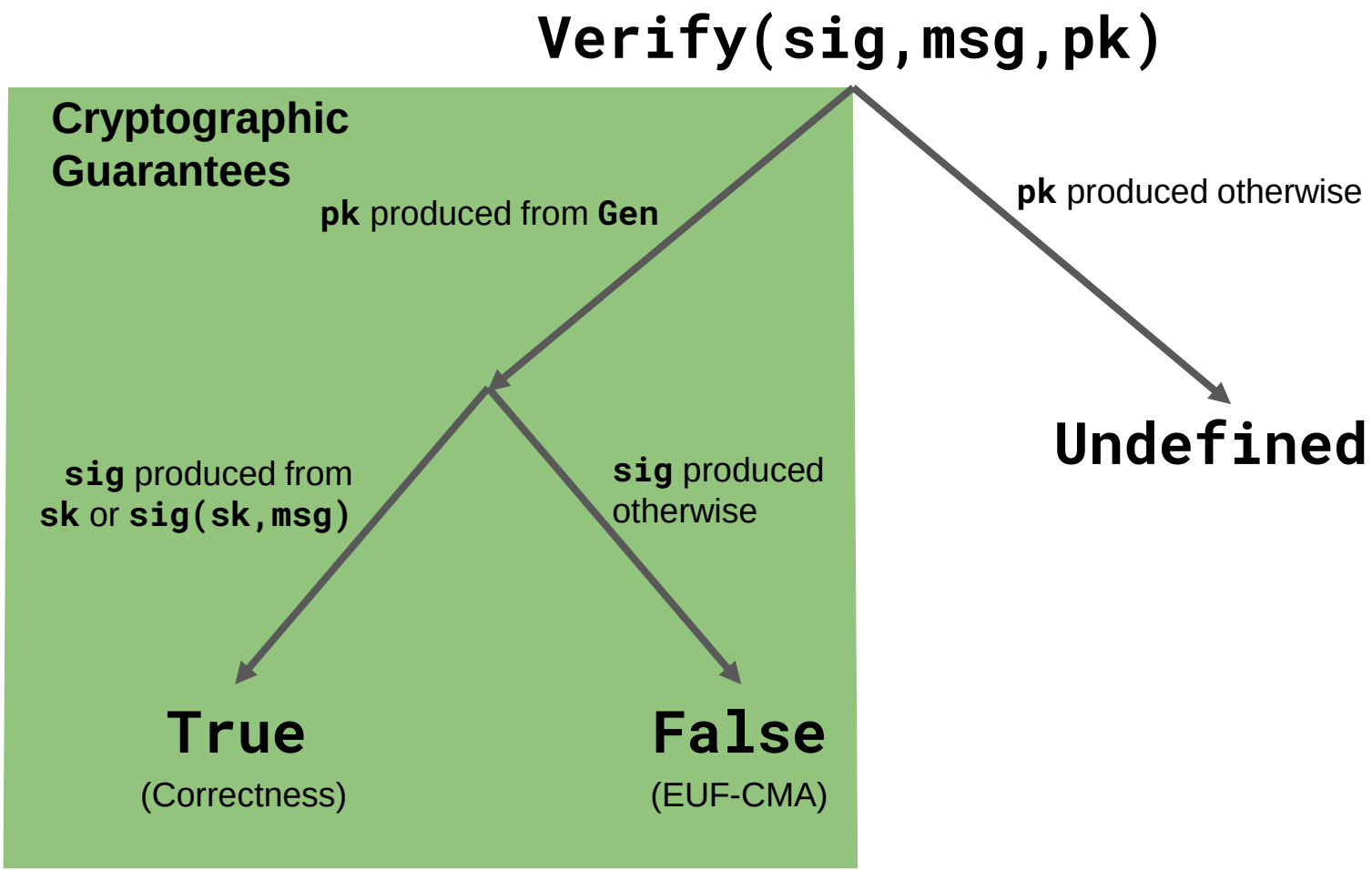
`Verify(sig,msg,pk)`



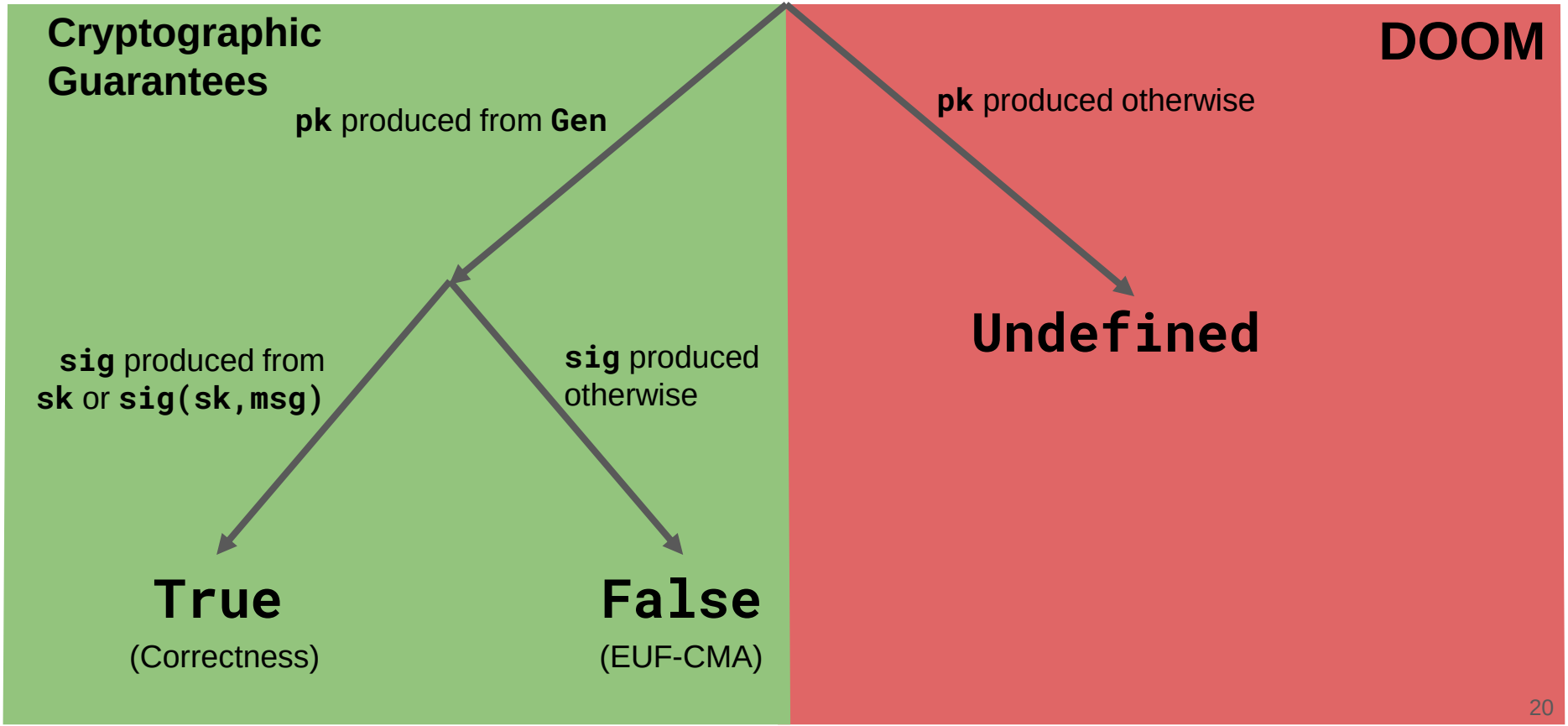

Verify(sig,msg,pk)
pk produced from Gen
pk produced otherwise
sig produced from sk or sig(sk,msg)
sig produced otherwise
True
(Correctness)


Verify(sig,msg,pk)
pk produced from Gen
pk produced otherwise
sig produced from sk or sig(sk,msg)
sig produced otherwise
True
(Correctness)
False
(EUF-CMA)


Verify(sig,msg,pk)
pk produced from Gen
pk produced otherwise
sig produced from sk or sig(sk,msg)
sig produced otherwise
Undefined
True
(Correctness)
False
(EUF-CMA)


Verify(sig,msg,pk)
Cryptographic Guarantees
pk produced from Gen
pk produced otherwise
Undefined
sig produced from sk or sig(sk,msg)
sig produced otherwise
True
(Correctness)
False
(EUF-CMA)

`Verify(sig,msg,pk)`



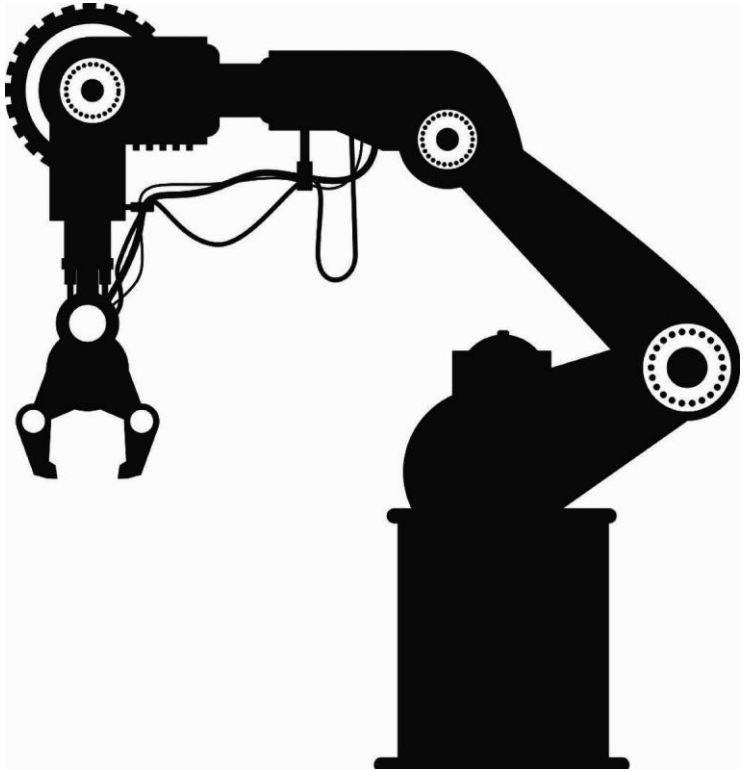
Prevalence

Proven Absent
Unknown
Present

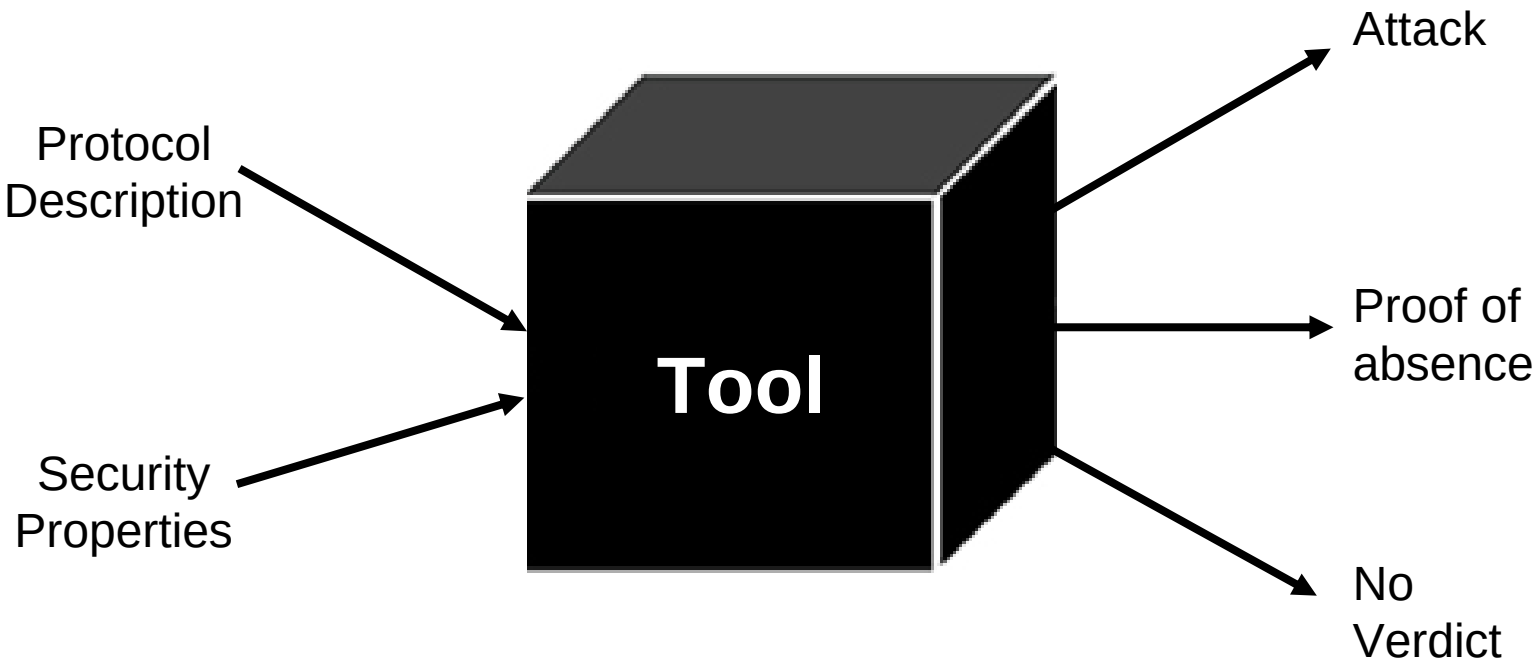
Signature scheme	KS	MKS	Coll.
RSA-PKCSv1.5	● [64]	● [64]	▲
RSA-PSS	● [64]	● [64]	▲
DSA	● [64]	● [64]	● [69]
ECDSA-FreeBP	● [26]	● [26]	● [67]
ECDSA-FixedBP	■ [59]	■ [59]	● [67]
Ed25519	■ [47]	■ [47]	● [19]
Ed25519-IETF	■ [47]	■ [47]	● [19]

[64] Pornin, T., & Stern, J. P. (2005). [26] Blake-Wilson, S., & Menezes, A. (1999). [59] Menezes, A., & Smart, N. (2001).
[47] Günther, F., & Poettering, B. (2017). [69] Vaudenay, S. (2003). [67] Stern, Jacques, et al. (2002) [19] Bernstein, Daniel J., et al (2012).

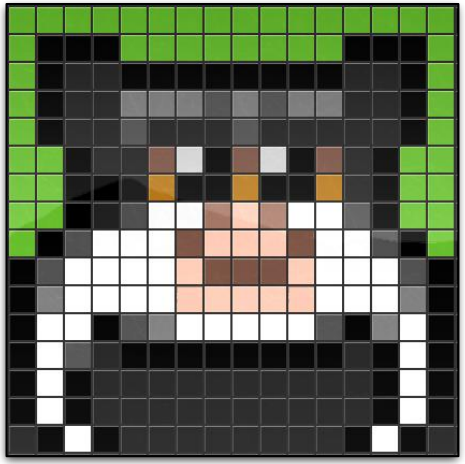
Automated Protocol Analysis



Automated Protocol Analysis

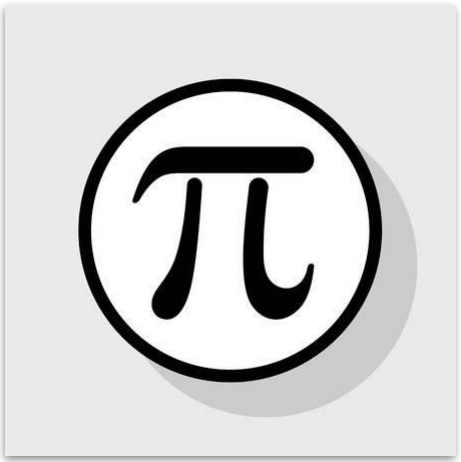


Tools for Automated Protocol Analysis



Tamarin

TLS1.3
ACME / Let's Encrypt
WebAuthN
Wireguard
Signal
Noise
GSM 5G
EMV Card Payments
V2X
IPSec / IKE



ProVerif

These tools have used the same signature model since **2001**

Our Enriched Models

Attack Finding Model

- Traditional Symbolic Model
- Functions represent explicit adversary capabilities
- Only captures known behaviours

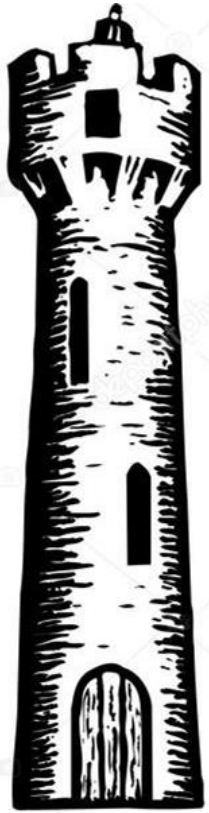
Verification Model

- Constraint Based Model
- Captures any permitted manipulation of verification
- Impractical attacks possible

Attacker can **only** perform explicitly **enumerated** actions

Attacker can do **anything** **except** violate constraints

Real World Attacks



New Attack on WS Security


```
<soap:Envelope ...>
 <soap:Header>
  <wsse:Security>
   <wsse:BinarySecurityToken wsu:Id="myKey" ...>
... security token ...
   </wsse:BinarySecurityToken>
   <sig:Signature>
    <sig:SignedInfo>
     <sig:Reference URI="#myMsg">...digest...</sig:Reference>
    </sig:SignedInfo>
... signature ...
    <sig:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI="#myKey"/>
     </wsse:SecurityTokenReference>
    </sig:KeyInfo>
   </sig:Signature>
  </wsse:Security>
 </soap:Header>
 <soap:Body wsu:Id="myMsg">
  <app:StockSymbol ...>
... application sub messages ...
  </app:StockSymbol>
 </soap:Body>
</soap:Envelope>
```

Black - SOAP elements
Red - WSS elements/attributes
Green - XML Signature elements

Academic Scrutiny of WS Security

Verified Reference Implementations of WS-Security Protocols

Microsoft Research

TulaFale: A Security Tool for Web Services

Microsoft Research

Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario

[1] IBM Zurich Research Lab, Switzerland
[2] Information Security Group, ETH Zurich, Switzerland

Secure Sessions for Web Services

ABSTRACT

WS-Security provides basic means to secure SOAP traffic, one envelope at a time. For typical web services, however, using WS-Security independently for each message is rather inefficient; besides, it is often important to secure the integrity of a whole session, as well as each message. To these ends, recent specifications provide further SOAP-level mechanisms. WS-SecureConversation introduces *security contexts*, which can be used to secure sessions between two parties. WS-Trust specifies how security contexts are issued and obtained.

We develop a semantics for the main mechanisms of WS-Trust and WS-SecureConversation, expressed as a library for TulaFale, a formal scripting language for security protocols. We model typical protocols relying on these mechanisms, and automatically prove their main security properties. We also informally discuss some limitations of these specifications.

Verified Interoperable Implementations of Security Protocols

ity of Pennsylvania

Protocol Code
Symbolic Crypto
Concrete Crypto
Platform (CLR)
Crypto .Net
Some Other Implementation (WSE)
Symbolic Testing
Interoperability (via SOAP)

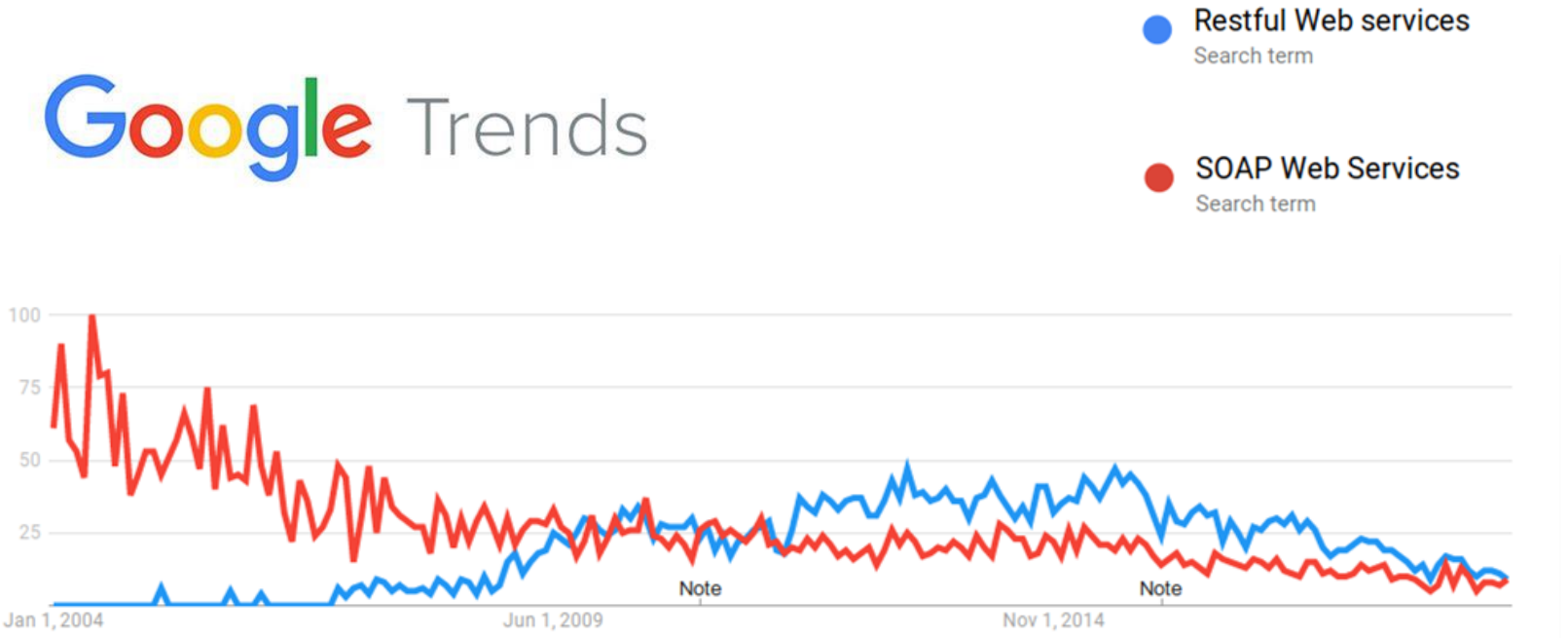
Verifying Policy-Based Security for Web Services

ABSTRACT

WS-SecurityPolicy is a declarative configuration language for driving web services security mechanisms. We describe a formal semantics for WS-SecurityPolicy, and propose a more abstract link language for specifying the security goals of web services and their

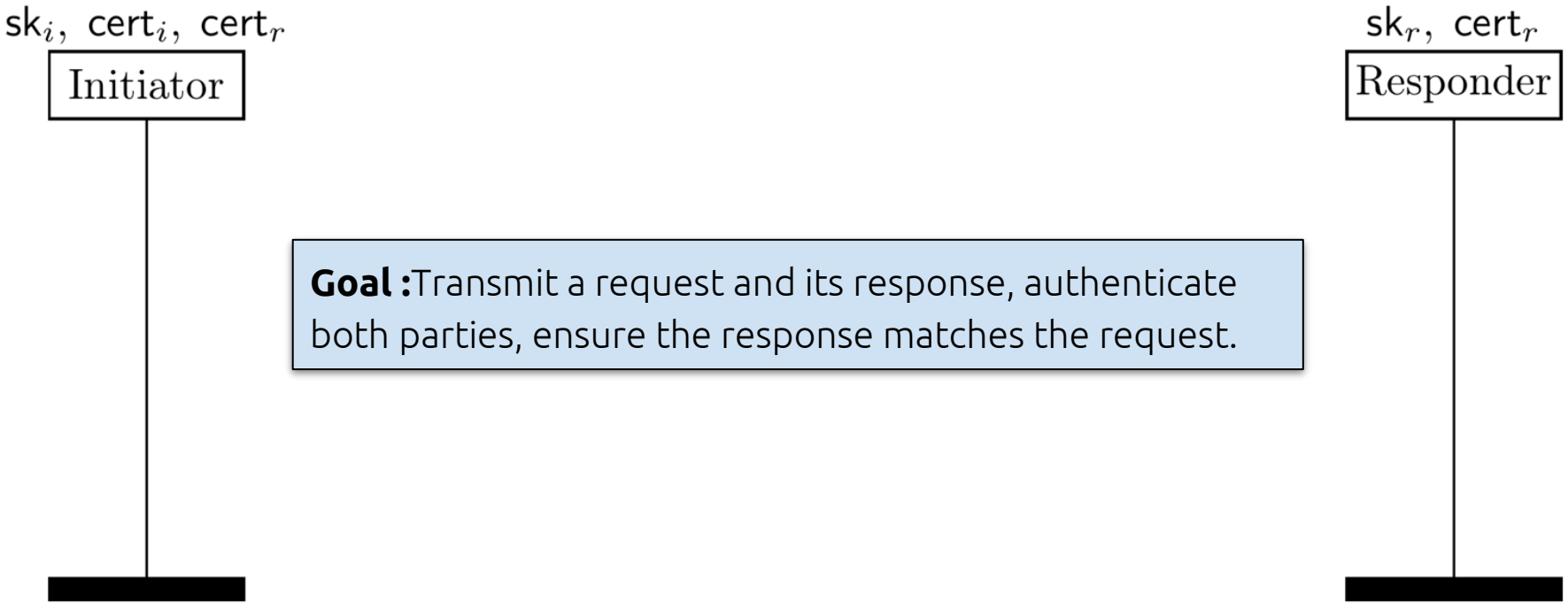
WS-SecurityPolicy [10], built on the WS-Policy [8] and WS-PolicyAssertion [9], is a declarative XML format for programming how web services implementations construct and check WS-Security headers. By expressing security checks as XML metadata instead of imperative code, policy-based web services conform to the general principle of isolating security checks from other aspects

WS Security Popularity



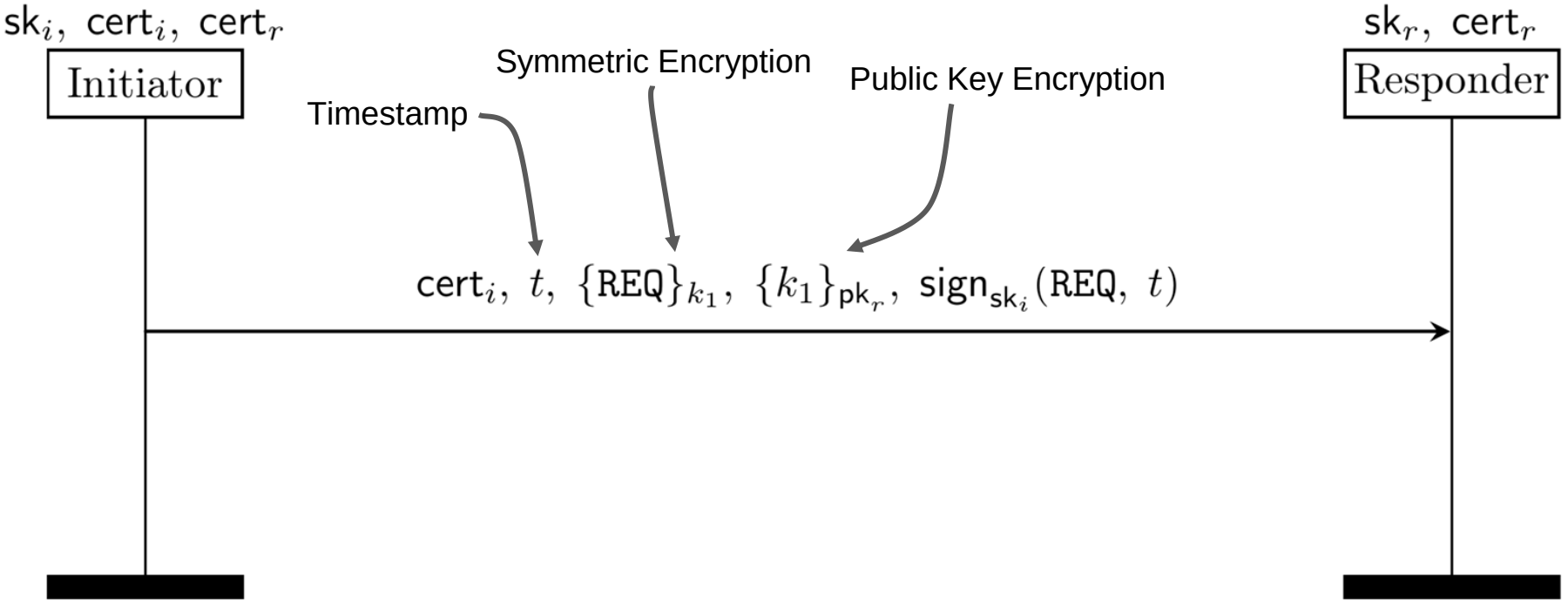
WS Security Support

WS Security X.509 Mutual Authentication

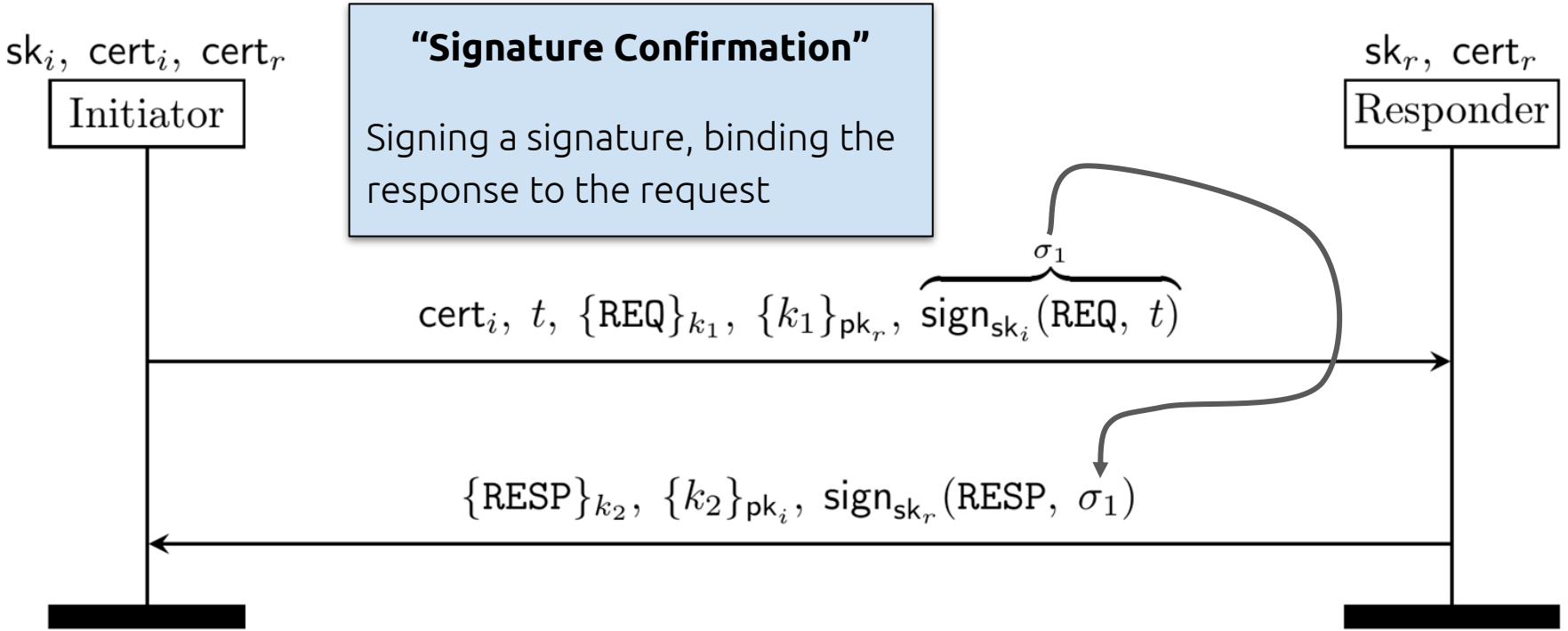


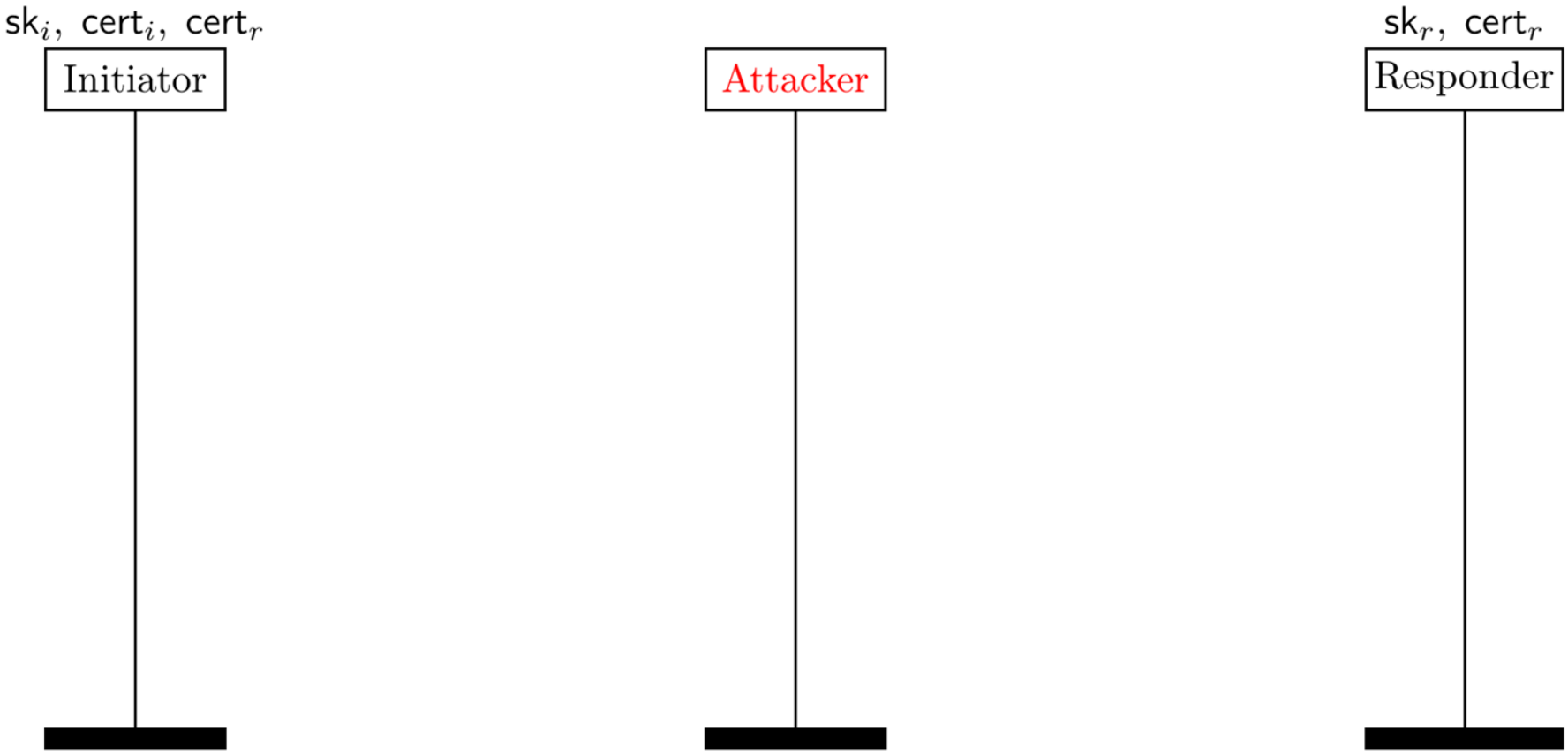
Goal : Transmit a request and its response, authenticate both parties, ensure the response matches the request.

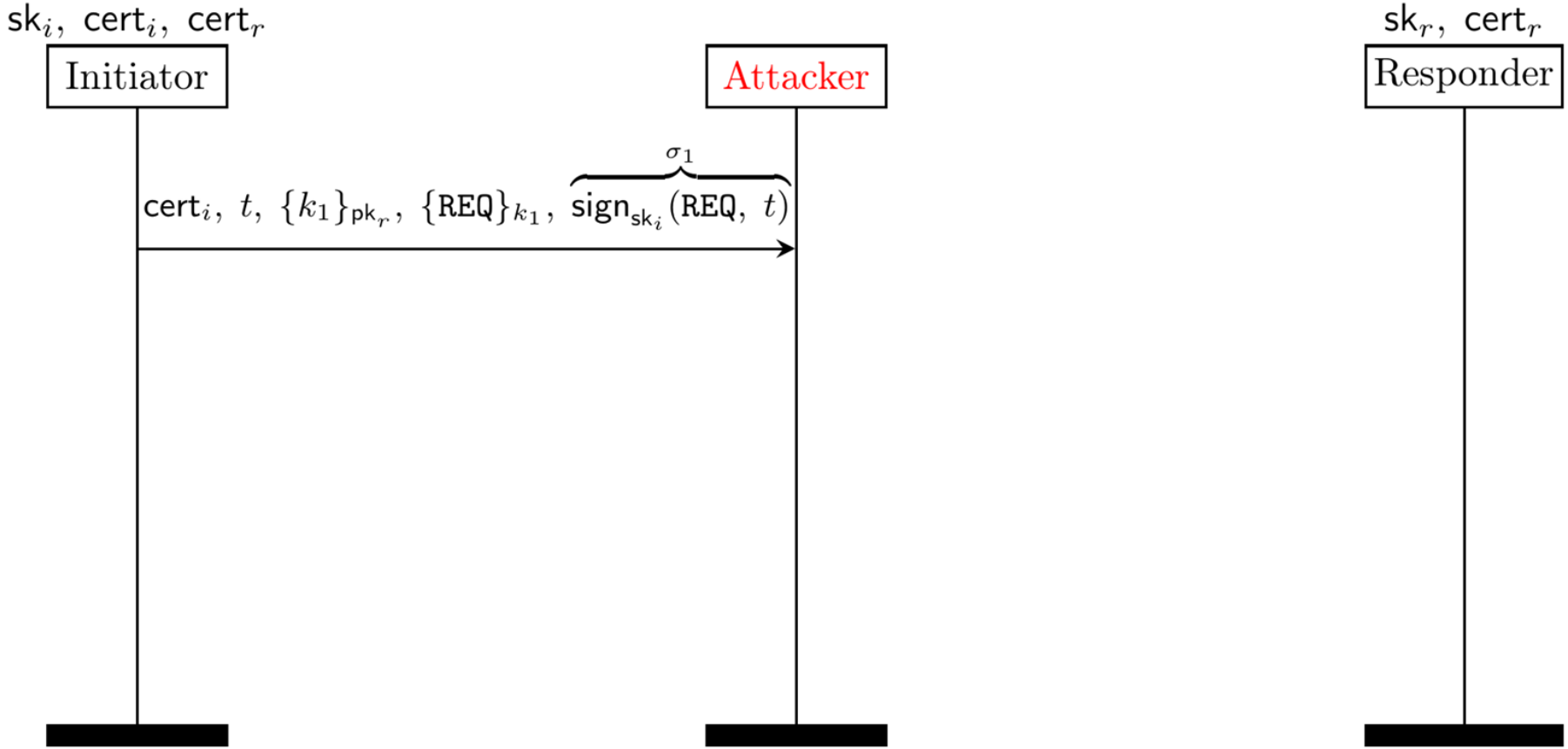
WS Security X.509 Mutual Authentication

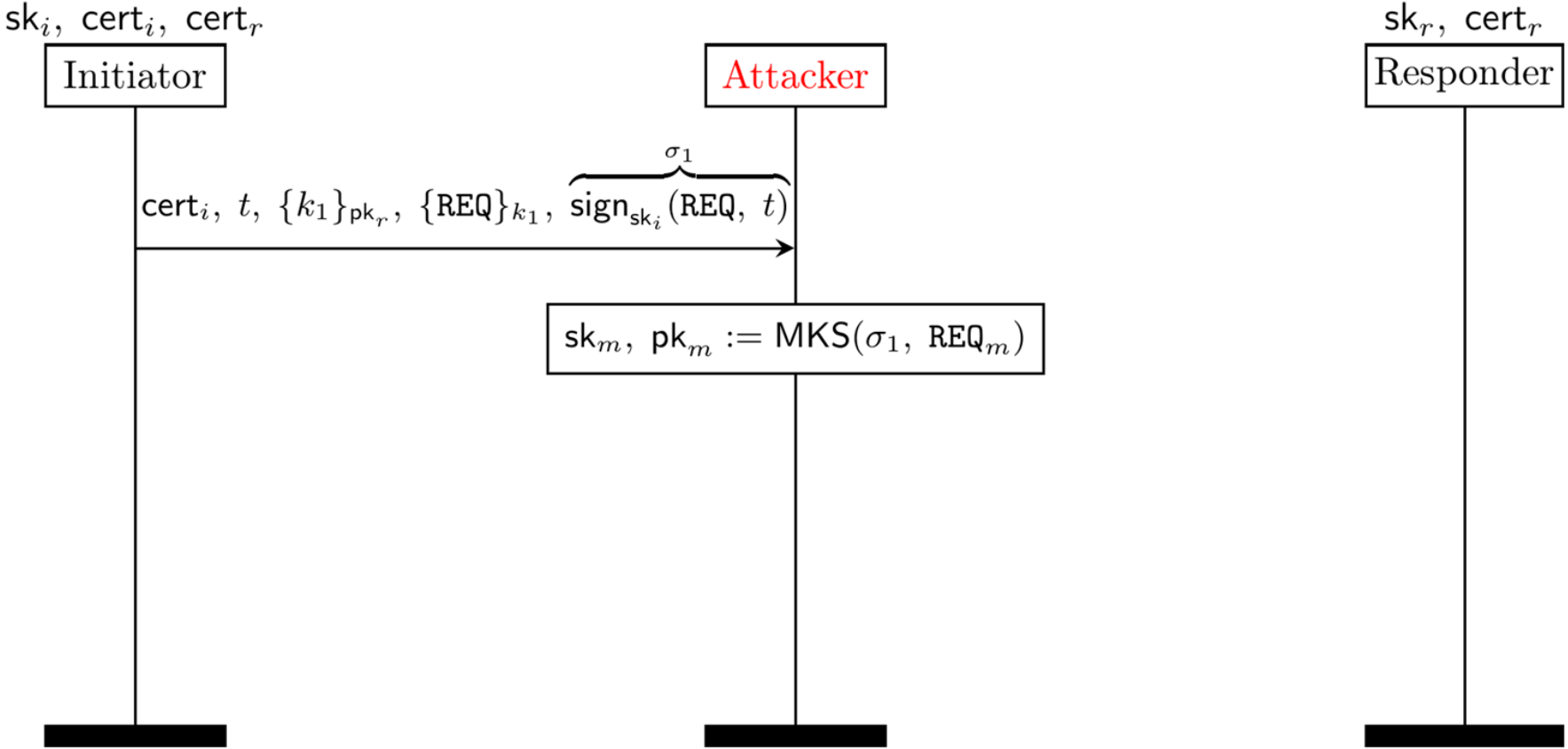


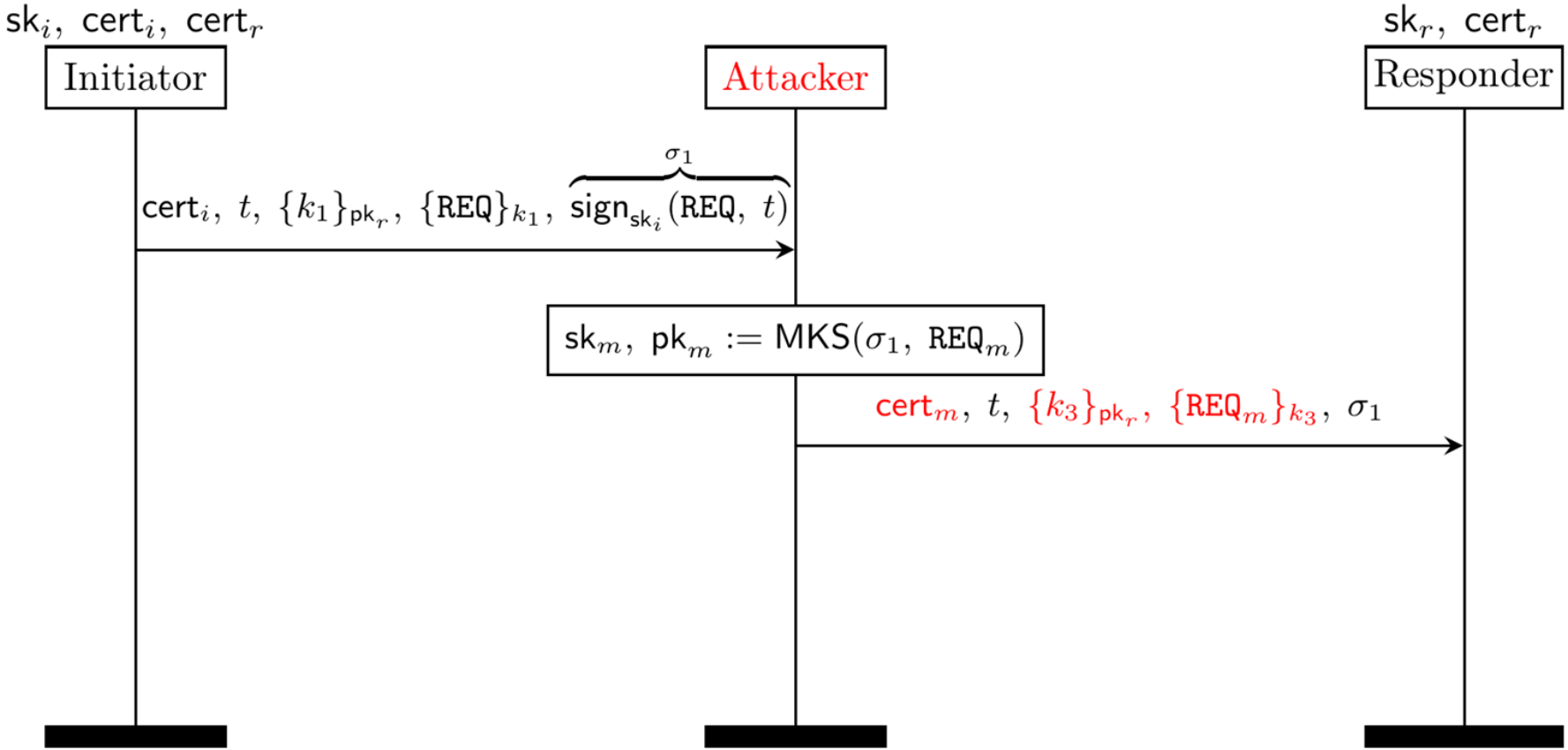
WS Security X.509 Mutual Authentication

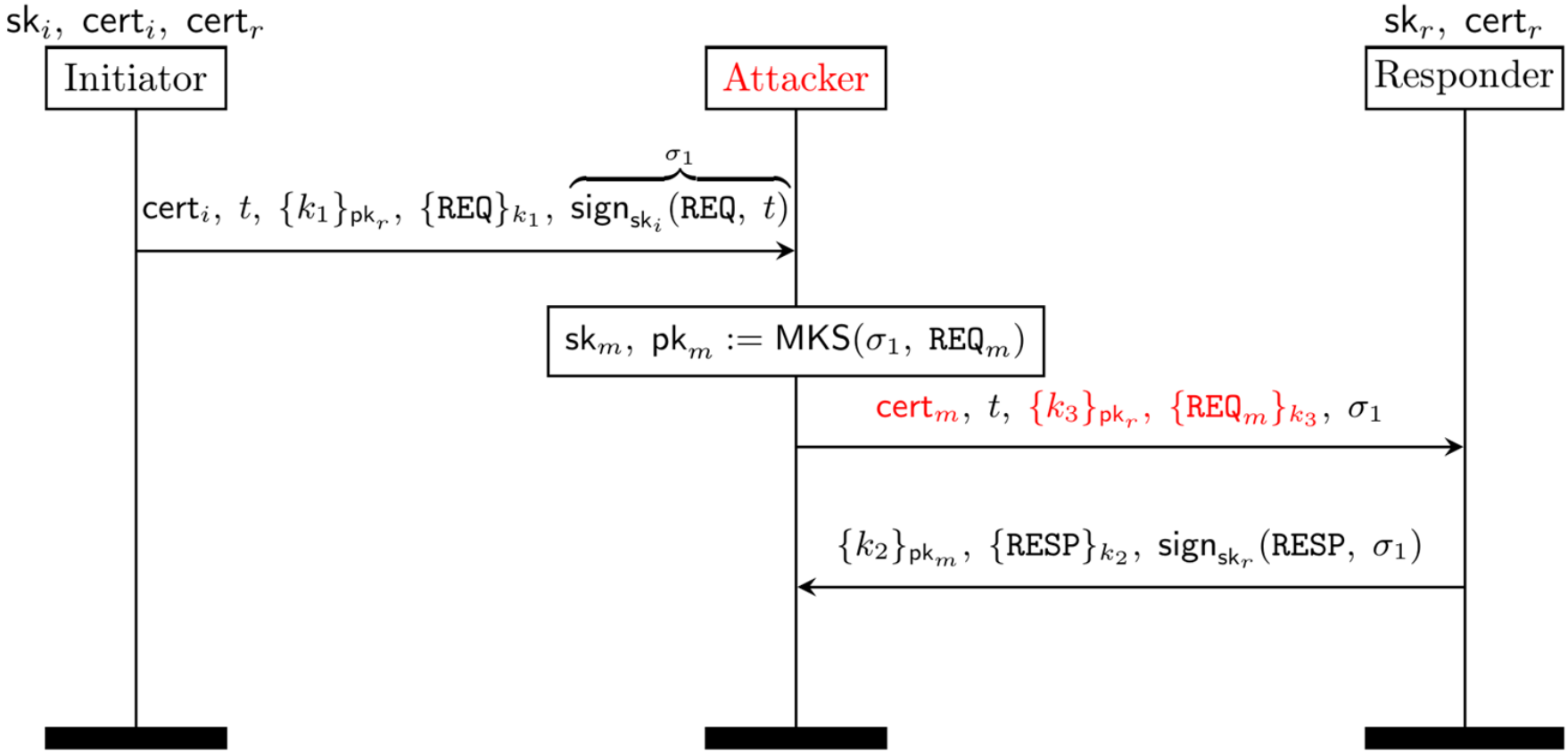


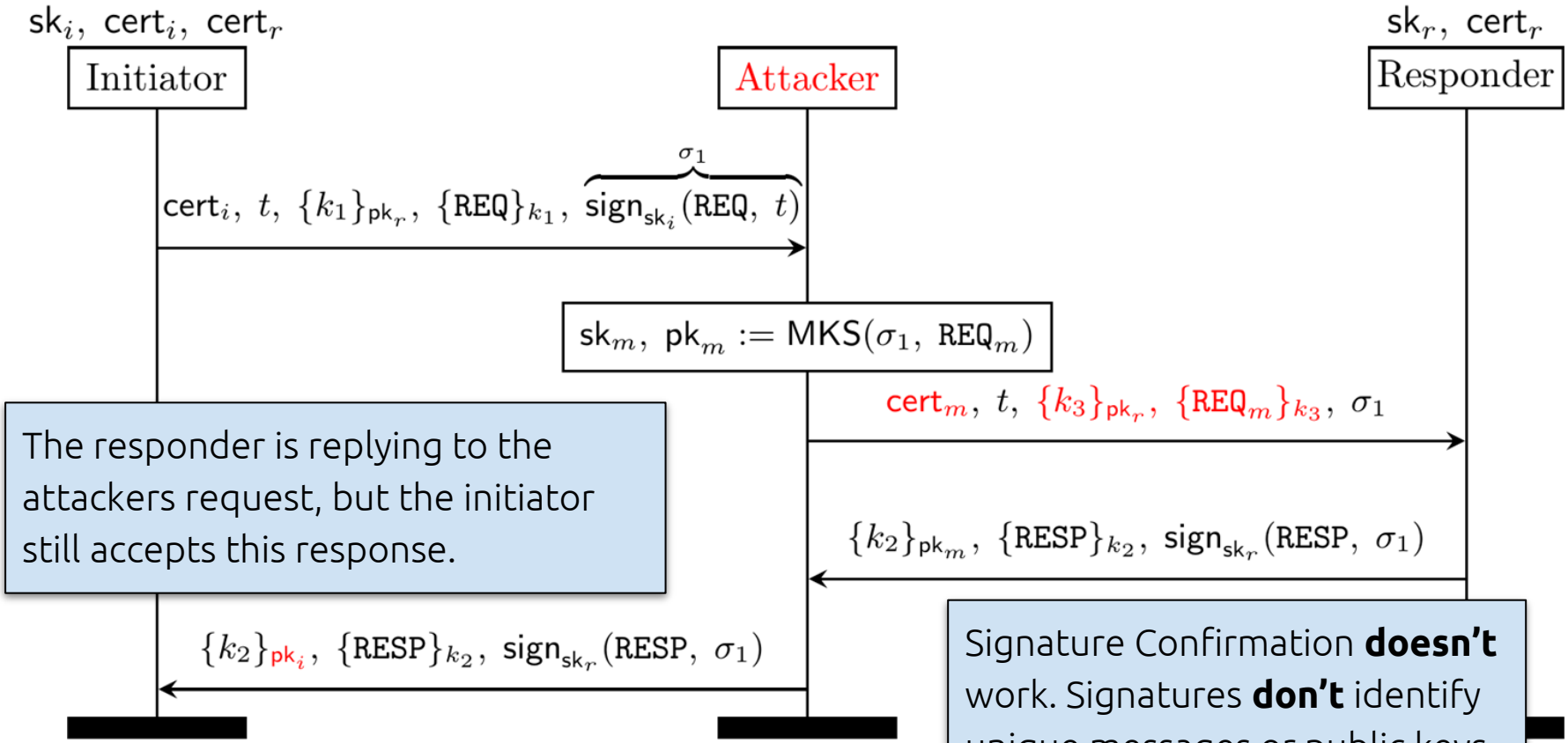

$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$
$\mathsf{cert}_m,\ t,\ \{k_3\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}_m\}_{k_3},\ \sigma_1$


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$
$\mathsf{cert}_m,\ t,\ \{k_3\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}_m\}_{k_3},\ \sigma_1$
$\{k_2\}_{\mathsf{pk}_m},\ \{\mathtt{RESP}\}_{k_2},\ \mathsf{sign}_{\mathsf{sk}_r}(\mathtt{RESP},\ \sigma_1)$


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$
$\mathsf{cert}_m,\ t,\ \{k_3\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}_m\}_{k_3},\ \sigma_1$
$\{k_2\}_{\mathsf{pk}_m},\ \{\mathtt{RESP}\}_{k_2},\ \mathsf{sign}_{\mathsf{sk}_r}(\mathtt{RESP},\ \sigma_1)$
$\{k_2\}_{\mathsf{pk}_i},\ \{\mathtt{RESP}\}_{k_2},\ \mathsf{sign}_{\mathsf{sk}_r}(\mathtt{RESP},\ \sigma_1)$
The responder is replying to the attackers request, but the initiator still accepts this response.
Signature Confirmation **doesn't** work. Signatures **don't** identify unique messages or public keys.

New Results

WS Security

ATTACK

12 seconds

DR Key

ATTACK

44 minutes

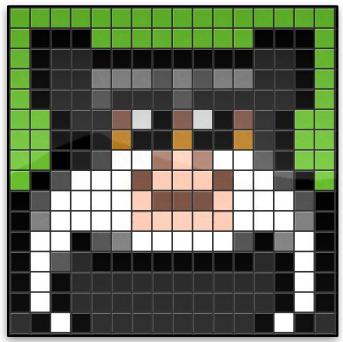
Scuttlebutt

ATTACK

2 minutes

Takeaways

- Signatures have some subtle and security critical behaviour
- Automated Protocol Analysis continues to mature
- New protocols, old attacks



Questions?

Key Substitution	Message-Key Substitution
Given a signature, produce a new public key which also verifies that signature	Given a signature, produce a new public key which also verifies that signature under *a different message.*
Colliding Signatures	**(Strong) Colliding Signatures**
Compute a public key and signature which verifies for more than one message	Compute a public key and signature which verifies for *any message*

Signature scheme	KS	MKS	Coll.
RSA-PKCSv1.5	● [64]	● [64]	▲
RSA-PSS	● [64]	● [64]	▲
DSA	● [64]	● [64]	● [69]
ECDSA-FreeBP	● [26]	● [26]	● [67]
ECDSA-FixedBP	■ [59]	■ [59]	● [67]
Ed25519	■ [47]	■ [47]	● [19]
Ed25519-IETF	■ [47]	■ [47]	● [19]

OASIS Open standards. Open source.

WS Security

ATTACK

12 seconds

SCION

DR Key

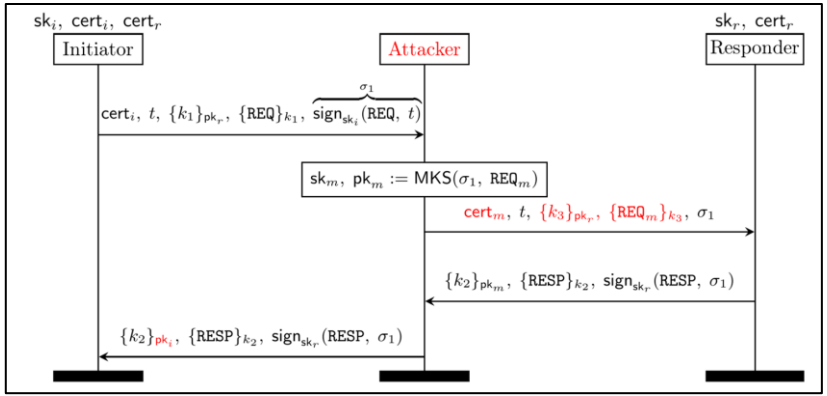
ATTACK

44 minutes

Scuttlebutt

ATTACK

2 minutes



@dennis__jackson

Previous Verifications

	Previous verification		Attacks found with our new signature models			
Protocol	**Year**	**Methodology**	**Properties violated**	**Model**	**Time (s)**	**First Reported**
X.509 Mutual Auth	2006	ProVerif	Correlation & Secrecy	no-CEO	5	*This paper*
WS Request-Response	2008	*F#* → ProVerif				
STS-MAC-fix1	2012	Tamarin	Authentication	no-CEO	35	[36]
STS-MAC-fix2	2012	Tamarin	Authentication	no-DEO	68	[23]
DRKey & OPT	2014	Coq	Authentication	Coll.	2640	*This paper*
ACME Draft 4	2017	ProVerif	DNS Validation	no-DEO	53	[6]

Note: Scuttlebutt was previously verified in Tamarin in a whitepaper which was not peer reviewed.

New Attacks

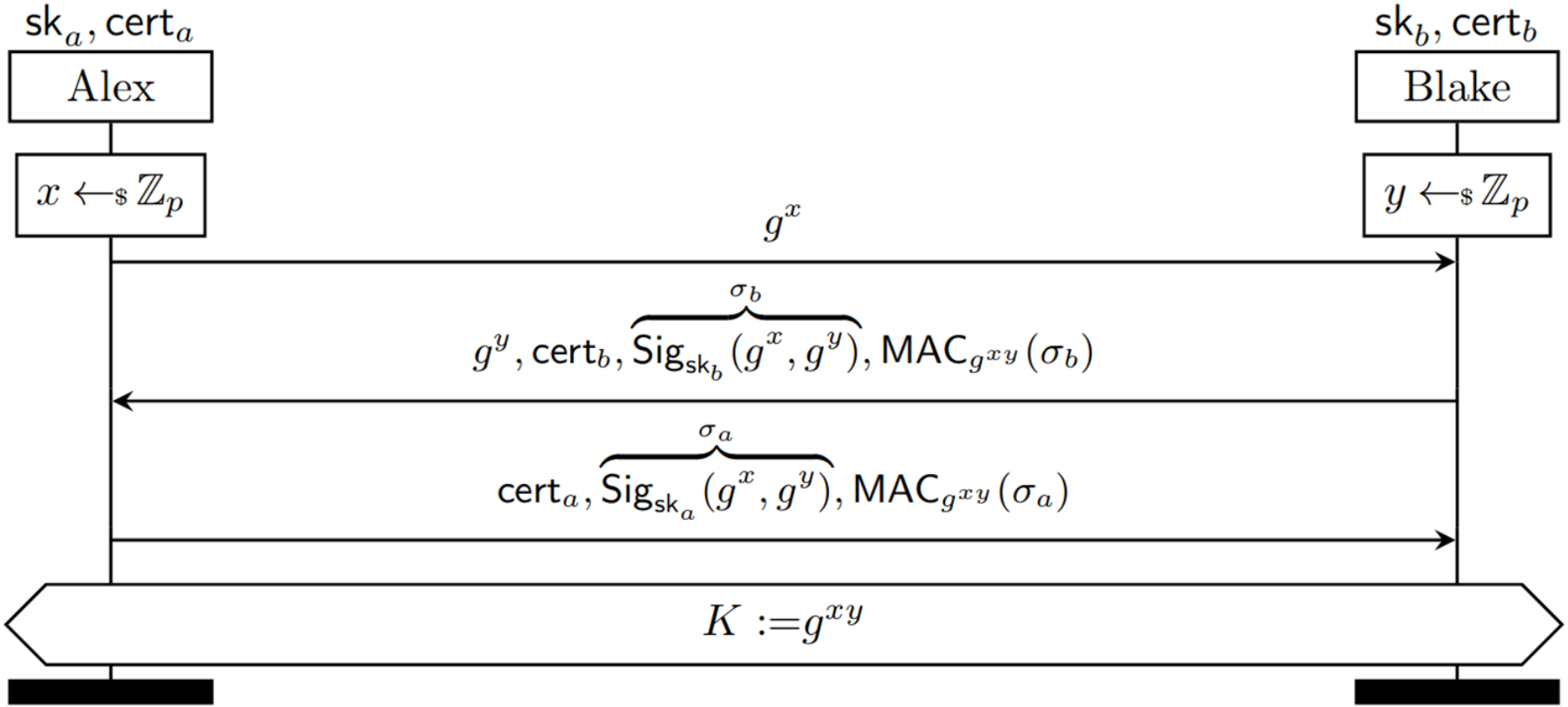
Protocol	Purpose	Impact	First Reported	Automatic Discovery
WS-Security X.509 Mutual Auth	Authenticate & Bind Messages	Loss of Authentication and secrecy	This work	12 seconds
DRKey	Key Distribution for Internet Routers	Redirection of network flows	This work	2640 seconds
Secure Scuttlebutt	Distributed Web / Social Network	Loss of authentication and secrecy	Companion Work[1]	131 seconds

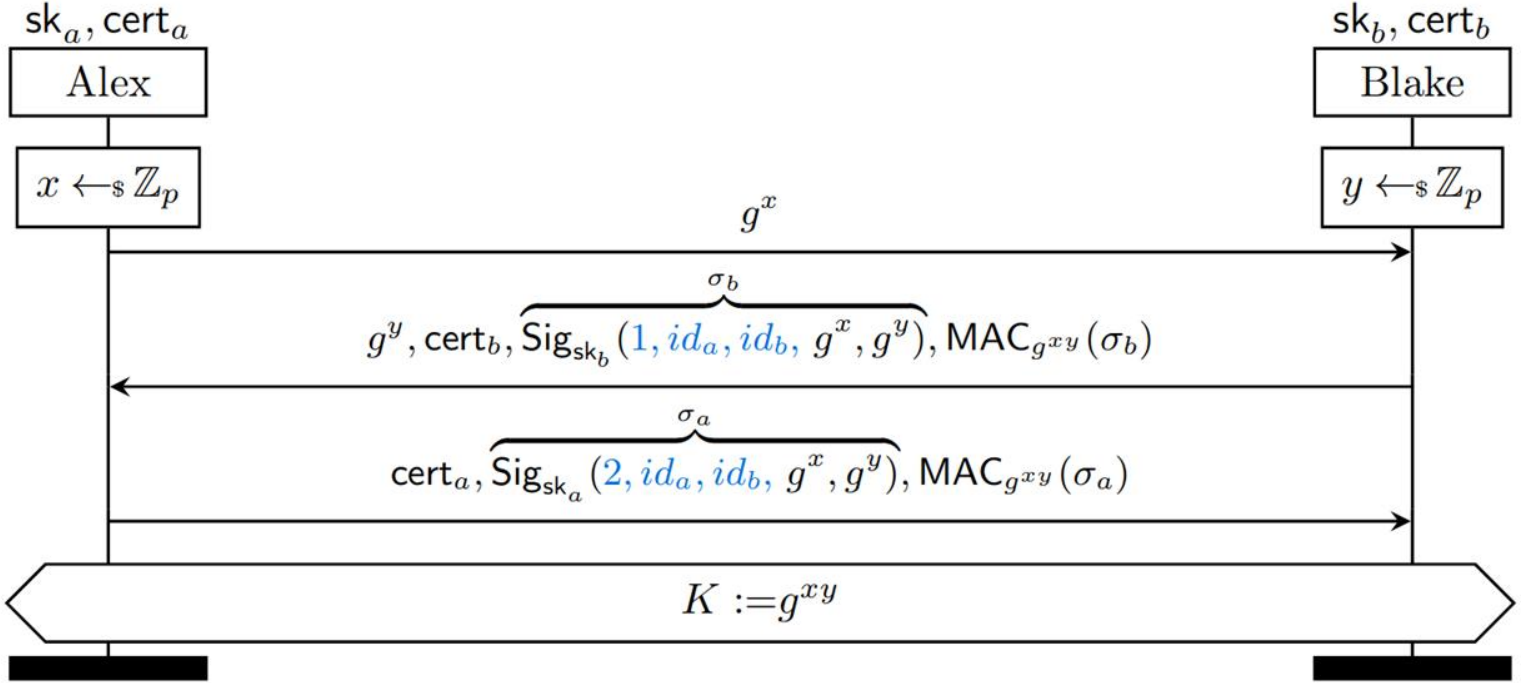
Cremers, C., & Jackson, D. (2019)

Previously Known Attacks

Protocol	Purpose	Impact	First Reported	Automatic (Re)discovery
Station to Station with MAC	Authenticated Key Exchange	Unknown Key Share Attack	1999[1]	23 seconds
Station to Station with MAC & ID	Authenticated Key Exchange	Unknown Key Share Attack	2000[2]	16 seconds
Let’s Encrypt (Draft 4)	DNS Challenge for TLS Cert	Website Impersonation	2015[3]	98 seconds

[1] Blake-Wilson, S., & Menezes, A. (1999). [2] Baek, J., & Kim, K. (2000). [3]Ayer, A. (2015)


$\mathsf{sk}_a, \mathsf{cert}_a$
Alex
$x \leftarrow_\$ \mathbb{Z}_p$
$\mathsf{sk}_b, \mathsf{cert}_b$
Blake
$y \leftarrow_\$ \mathbb{Z}_p$
g^x
$g^y, \mathsf{cert}_b, \overbrace{\mathsf{Sig}_{\mathsf{sk}_b}(g^x, g^y)}^{\sigma_b}, \mathsf{MAC}_{g^{xy}}(\sigma_b)$
$\mathsf{cert}_a, \overbrace{\mathsf{Sig}_{\mathsf{sk}_a}(g^x, g^y)}^{\sigma_a}, \mathsf{MAC}_{g^{xy}}(\sigma_a)$
$K := g^{xy}$


$\mathsf{sk}_a, \mathsf{cert}_a$
Alex
$x \leftarrow_\$ \mathbb{Z}_p$
$\mathsf{sk}_b, \mathsf{cert}_b$
Blake
$y \leftarrow_\$ \mathbb{Z}_p$
g^x
σ_b
$g^y, \mathsf{cert}_b, \overbrace{\mathsf{Sig}_{\mathsf{sk}_b}(1, id_a, id_b, g^x, g^y)}, \mathsf{MAC}_{g^{xy}}(\sigma_b)$
σ_a
$\mathsf{cert}_a, \overbrace{\mathsf{Sig}_{\mathsf{sk}_a}(2, id_a, id_b, g^x, g^y)}, \mathsf{MAC}_{g^{xy}}(\sigma_a)$
$K := g^{xy}$