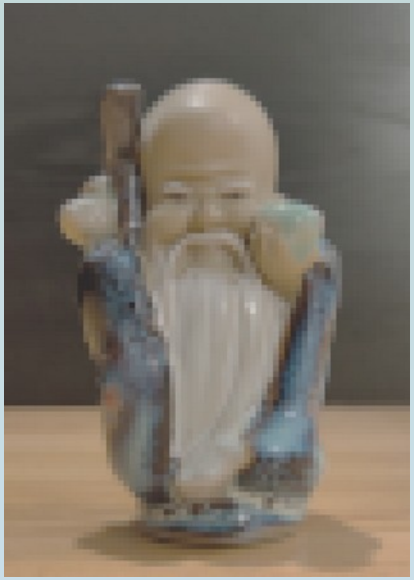
Tamarin tutorial afternoon:

More accurate/larger models

Cas Cremers

Research directions

Reality



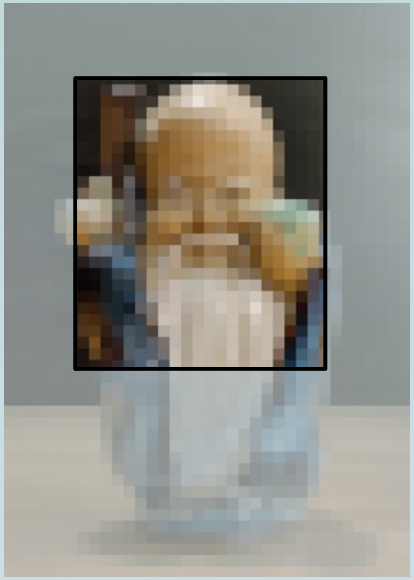
Computational

Symbolic

Research directions

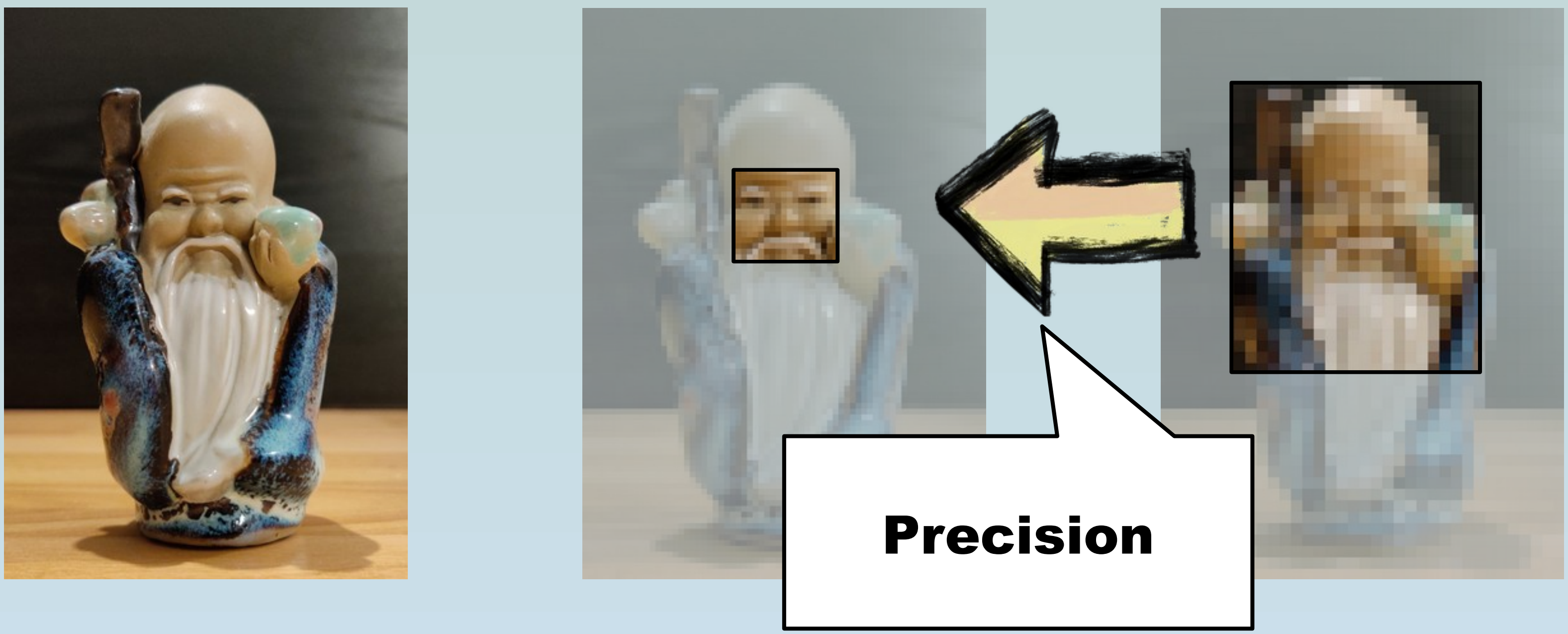
Reality

Computational



Symbolic

Research directions



Reality Computational Symbolic

Research directions

Reality

Computational

Sym… c

Precision

Scaling and automation

Two things that stuck in the back of my head

Around 2006: Duplicate Signature Key Selection (DSKS) attacks

Given any (e.g. RSA) signature, you can create a second key pair whose verification key also verifies that same signature??
(Related: unique ownership)

Around 2014: Small subgroups

Diffie-Hellman protocols expect to receive an element of a prime order group, but often don't check this. *This is usually not a problem?*
Bharghavan et. al. make a basic model in ProVerif for channel bindings work.

2016

Let's write a paper!

"Better Dolev-Yao abstractions of cryptographic primitives"

Plan:

- Revisit all Dolev-Yao primitives (signatures, exponentiation, encryption)
- Make better versions
- Submit
- Profit!!

Let's start with the easiest thing, **signatures**

2017

Let's write a paper!

~~***"Better Dolev-Yao abstractions of cryptographic primitives"***~~

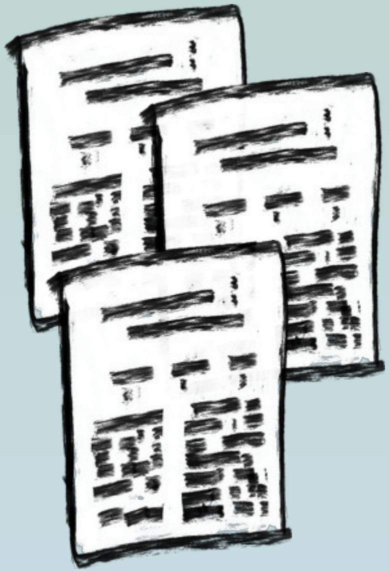
After months of work:

signatures alone are a paper

2017

Let's write ***three*** *papers!*

"Signatures"
"Diffie-Hellman"
"Authenticated Encryption"



Signatures

History of subtle signature properties

1999: Key Substitution [Blake-Wilson, Menezes]

Given **sig**, **pk**, and **msg**:
Calculate (**sk'**,**pk'**) such that (**sig**,**msg**,**pk'**) verifies

History of subtle signature properties

1999: Key Substitution [Blake-Wilson, Menezes]

Given **sig**, **pk**, and **msg**:
Calculate (**sk’**,**pk’**) such that (**sig**,**msg**,**pk’**) verifies

2000: Message-key Substitution [Baek, Kim]

Given **sig**,**pk**,**msg**, and **msg’**:

Calculate (**sk’**,**pk’**) such that (**sig**,**msg’**,**pk’**) verifies

Traditional Symbolic Signatures

`verify/2, sign/2, pk/1`

The Signature

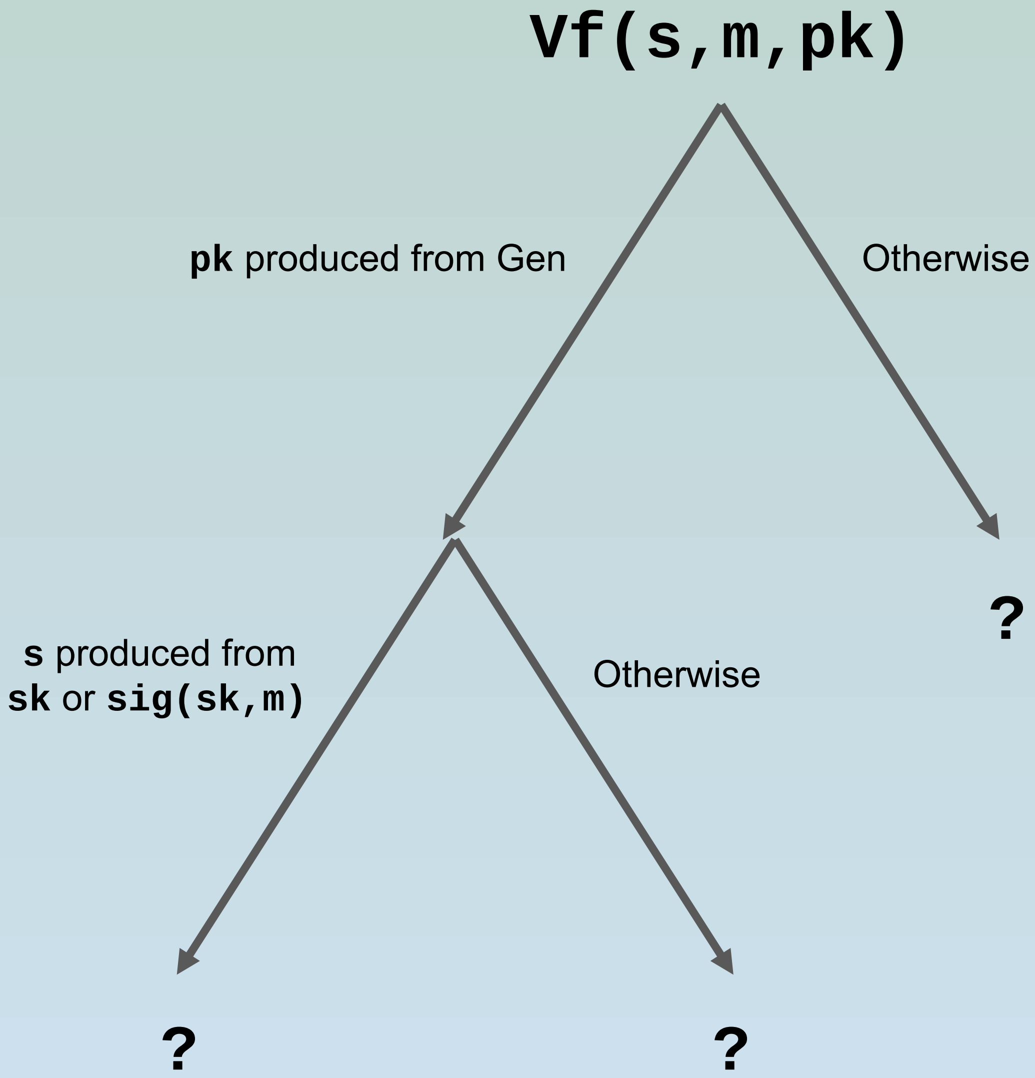
The Signer

```
verify(sign(A,DATA),DATA,pk(A)) = true
```

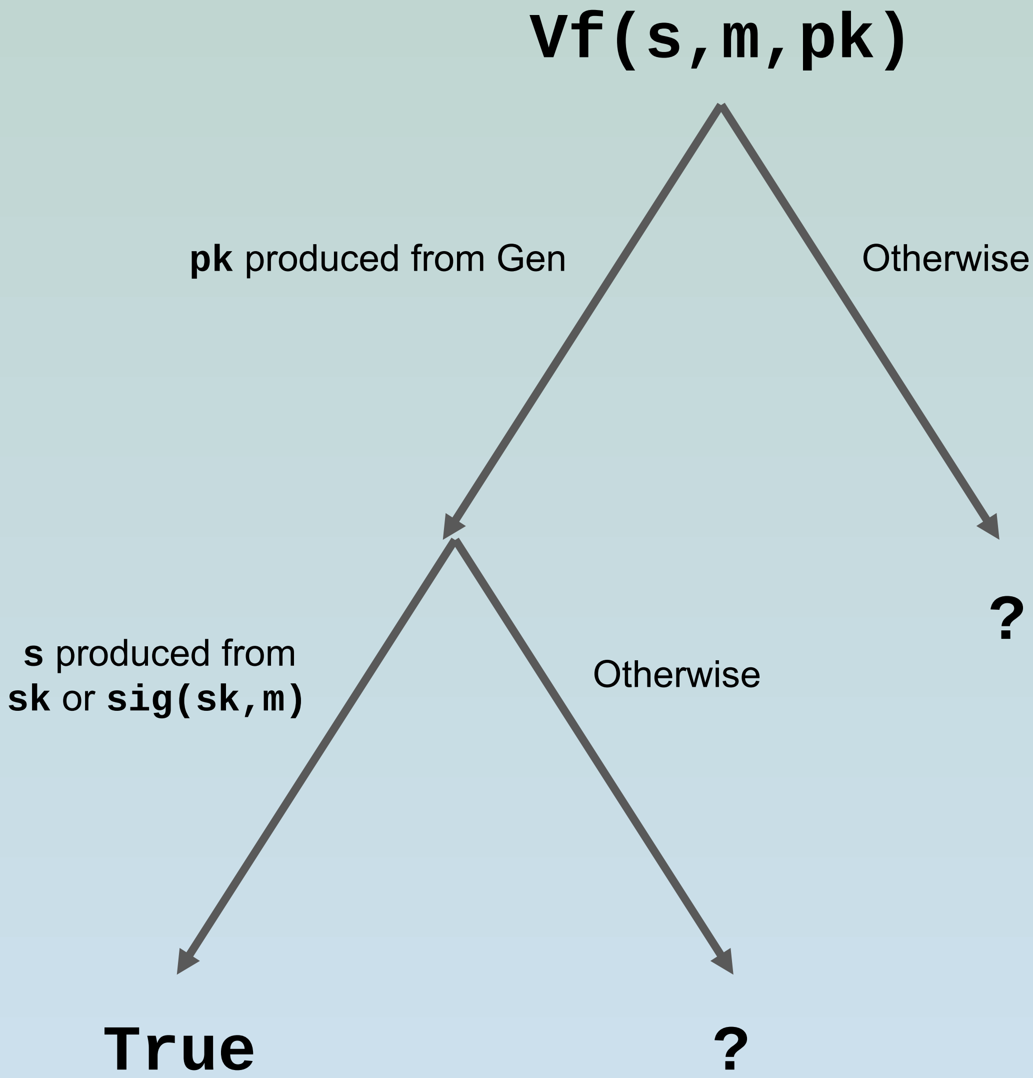
The Message

The Result

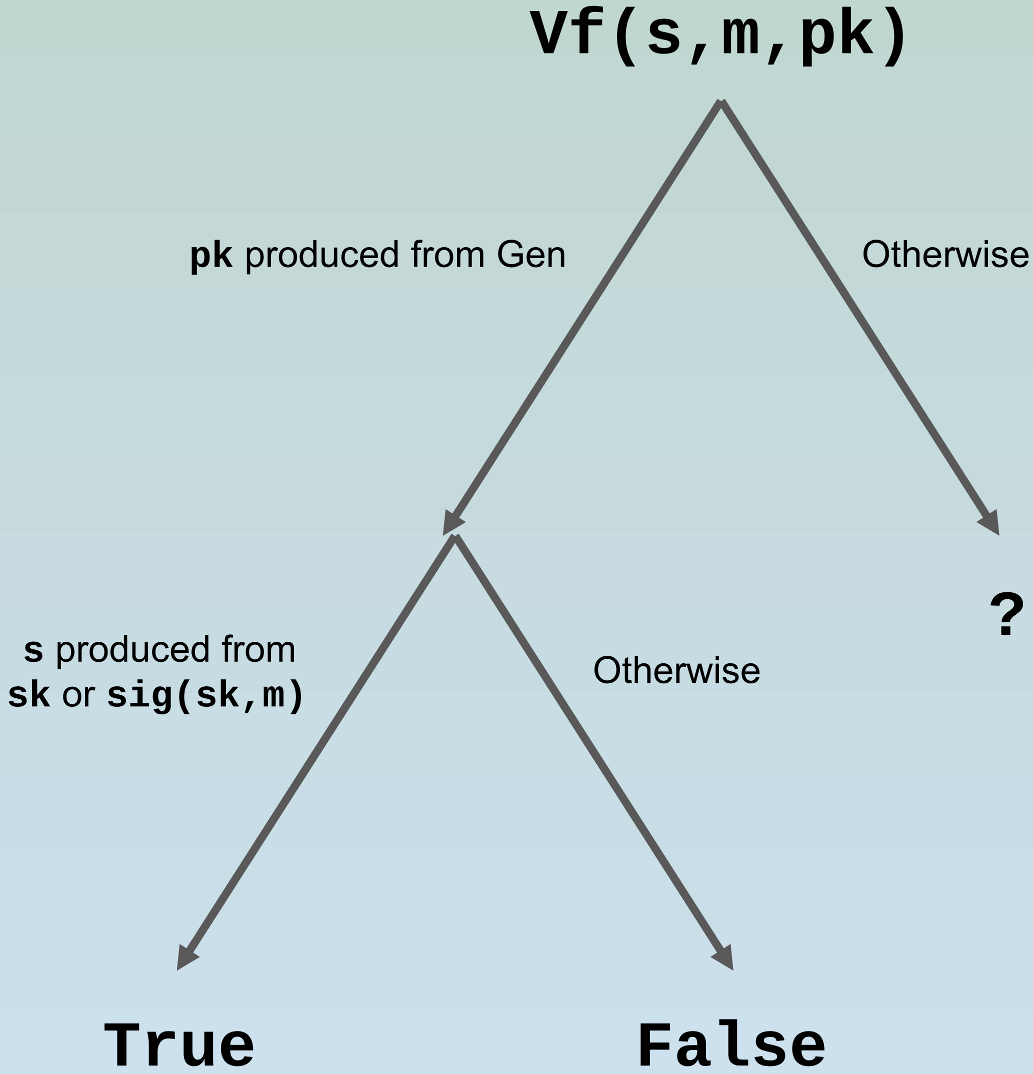
First published in 2001, used by all contemporary tools



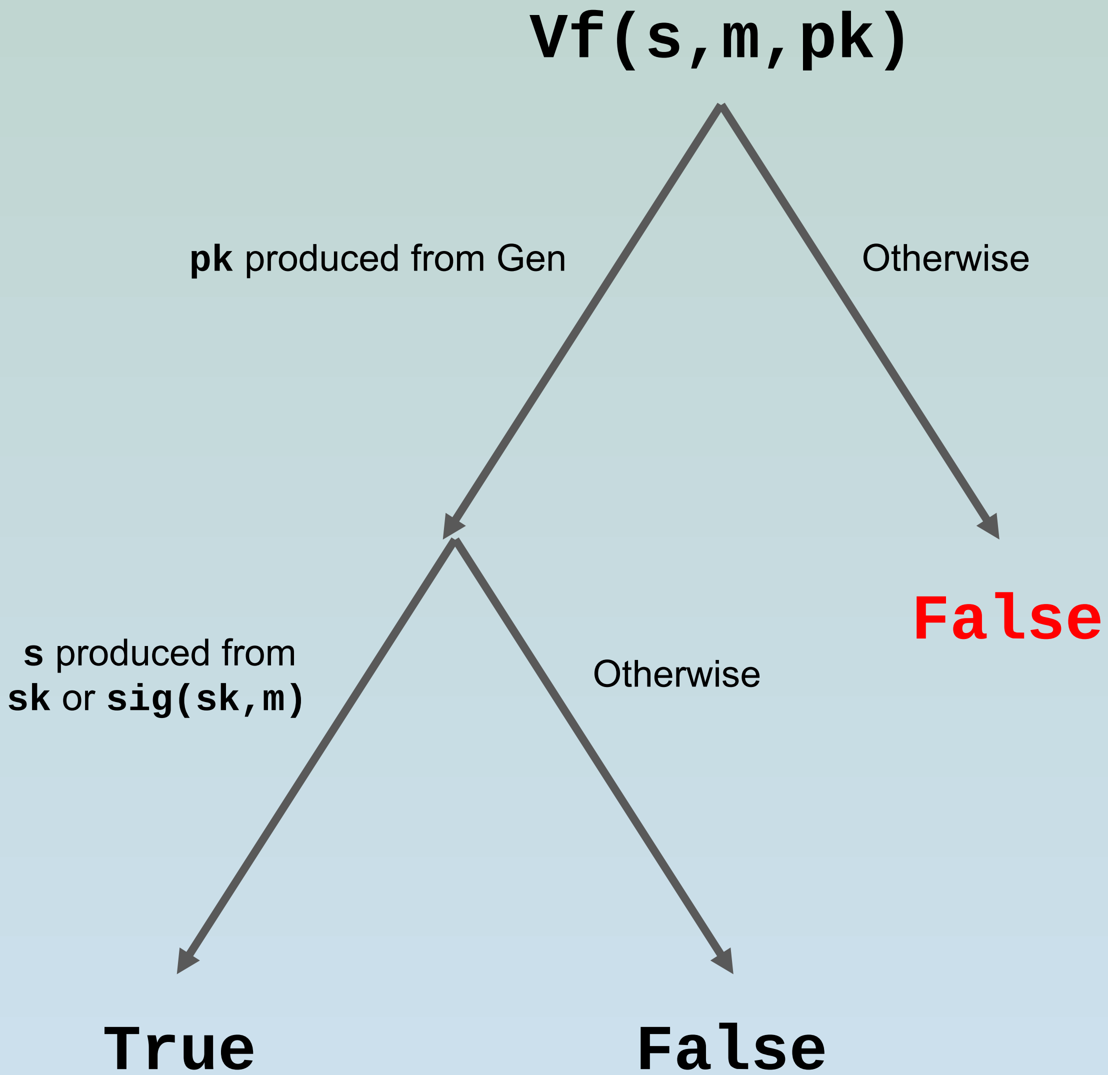
verify(sign(A,DATA),DATA,pk(A)) = true



`verify(sign(A,DATA),DATA,pk(A)) = true`



`verify(sign(A,DATA),DATA,pk(A)) = true`



`verify(sign(A,DATA),DATA,pk(A)) = true`


MIND THE GAP

No Conservative Exclusive Ownership

Given **`s,pk,m`** with

```
verify(s,m,pk) = true
```

Calculate **`sk’,pk’`** such that

```
verify(s,m,pk’) = true
```

First Reported: 1999 (as DSKS)

Applies to: RSA-PKCSv1.5, RSS-PSS, DSA, ECDSA with Free BP

No Destructive Exclusive Ownership

Given **`s,pk,m,m’`** with

`verify(s,m,pk) = true`

Calculate **`sk’,pk’`** such that

`verify(s,m’,pk’) = true`

First Reported: 2005

Applies to: RSA-PKCSv1.5, RSS-PSS, DSA, ECDSA with Free BP

Colliding

Given **`m,m’,`** calculate **`sk,pk,s`** such that

```
verify(s,m ,pk) = true
verify(s,m’,pk) = true
```

Reported: 2002

Applies to: ECDSA, Ed25519

Re-Signing

Given **`s,pk`** and **`sk’,pk’`** with

```
verify(s,m,pk) = true
```

Calculate **`s’`** such that

```
verify(s’,m,pk’) = true
```

Applies to: RSA-PKCSv1.5, RSA-PSS

Malleability

Given **`s,pk,m`** with

`verify(s,m,pk) = true`

Calculate **`s'`** such that

`verify(s',m,pk) = true`

Reported: 2002

Applies to: ECDSA, Ed25519

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]

Simplified table from [JCCS2019] ACM CCS 2019: *Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures*

[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).

Improving the Symbolic Model

Re-signing

```
resign(sign(m,sk1),sk2) = sign(m,sk2)
```

Malleability

```
mutate(sign(m,r1,sk),r2)) = sign(m,r2,sk)
```

Improving the Symbolic Model

CEO:

```
verify(sign(m,sk),m,pk(CEOgen(sign(m,sk)))) = true
```

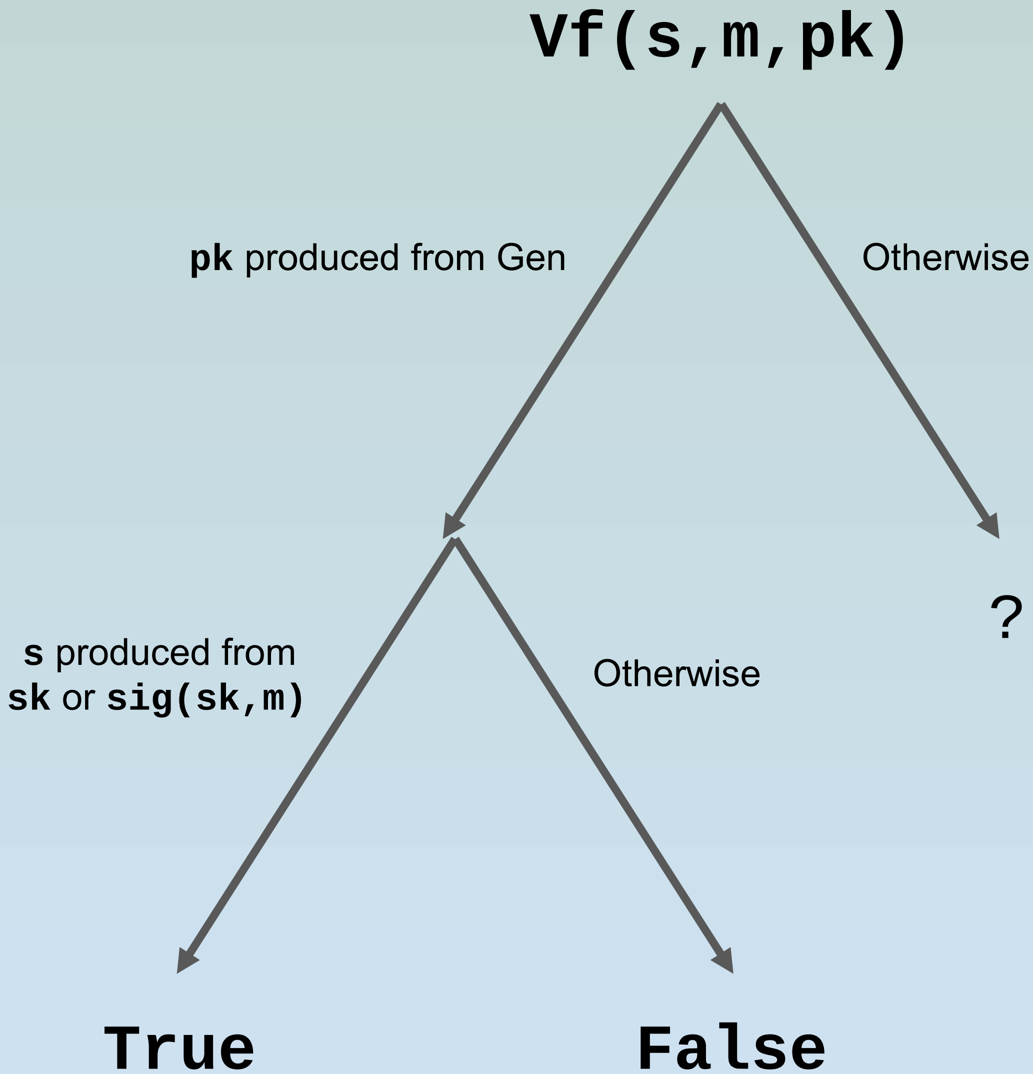
DEO:

```
verify(sign(m1,sk),m2,pk(DEOgen(sign(m1,sk),m2))) = true
```

Colliding:

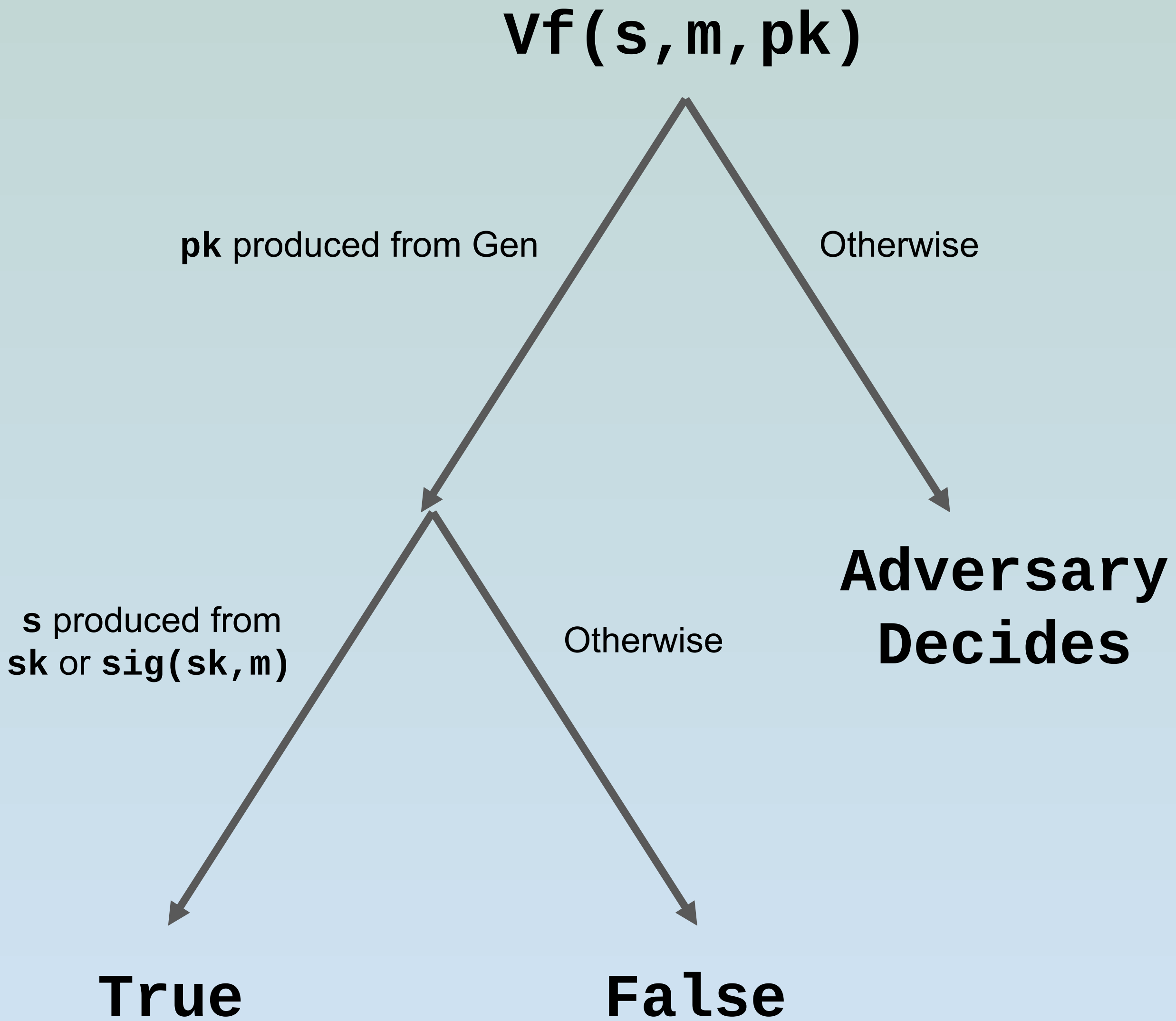
```
verify(sign(n,x),m,pk(weak(x))) = true
```

A Better Way?



Vf(s,m,pk)
pk produced from Gen
Otherwise
?
s produced from
sk or sig(sk,m)
Otherwise
True
False

A Better Way?



Vf(s,m,pk)
pk produced from Gen
Otherwise
s produced from
sk or sig(sk,m)
Otherwise
Adversary
Decides
True
False

Restrictions

- A protocol is made of steps
- Restrictions prevent a step from “triggering”
- Guarded Fragment of First Order Logic with Timepoints
- Only act on terms, not subterms

Examples:

- `∀ x,y Eq  (x,y) => x  = y`
- `∀ x,y InEq(x,y) => x != y`
- ∀ t_1,t_2 `OnlyOnce() @` t_1 `& OnlyOnce() @` t_2 `=>` t_1 `=` t_2

Lifting from Terms to Traces

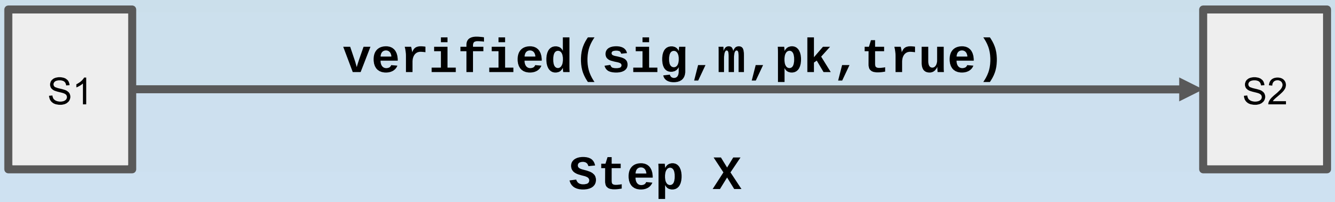
We remove **`verify`** and introduce new *step labels*:

`verified(sig,m,pk,result),` **`result ∈ {true,false}`**

`honest(pk)`

Any step where an honest party generates a public key, we label it with ‘honest.’

Now we can use *restrictions* to control when the ‘verified’ event can occur.



Restrictions

Correctness:

```
Honest(pk(a)) & Verified(sign(m,r,a),m,pk(a),False) => ⊥
```

Restrictions

Correctness:

```
Honest(pk(a)) & Verified(sign(m,r,a),m,pk(a),False) => ⊥
```

Unforgeability:

```
Honest(pk(a)) & Verified(s,m,pk(a),true) => s = sign(m,r,a)
```

Restrictions

Correctness:

`Honest(pk(a)) & Verified(sign(m,r,a),m,pk(a),False) =>` $\bot$

Unforgeability:

`Honest(pk(a)) & Verified(s,m,pk(a),true) => s = sign(m,r,a)`

Consistency:

`Verified(s,m,pk(a),r1) & Verified(s,m,pk(a),r2) => r1 = r2`

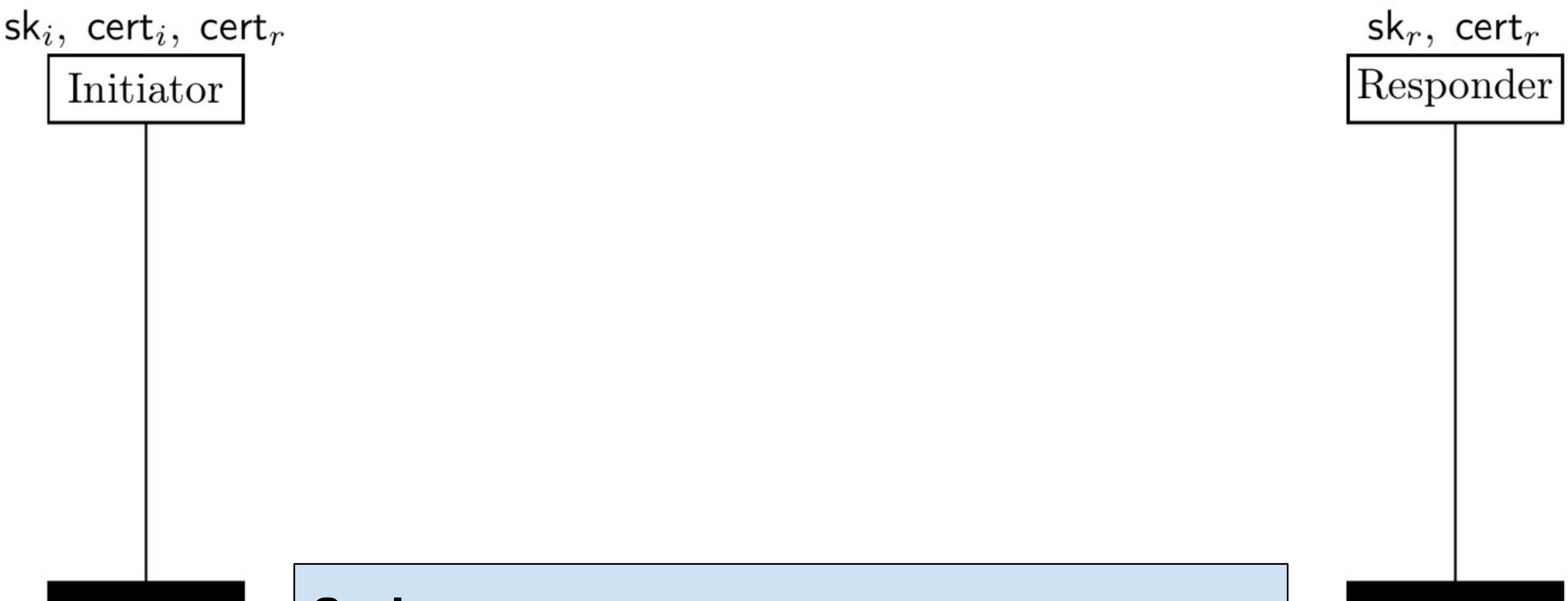
Case studies

Protocol	Previous verification	
	Year	Methodology
X.509 Mutual Auth	2006	ProVerif
WS Request-Response	2008	F# → ProVerif
STS-MAC-fix1	2012	Tamarin
STS-MAC-fix2	2012	Tamarin
DRKey & OPT	2014	Coq
ACME Draft 4	2017	ProVerif

Case studies

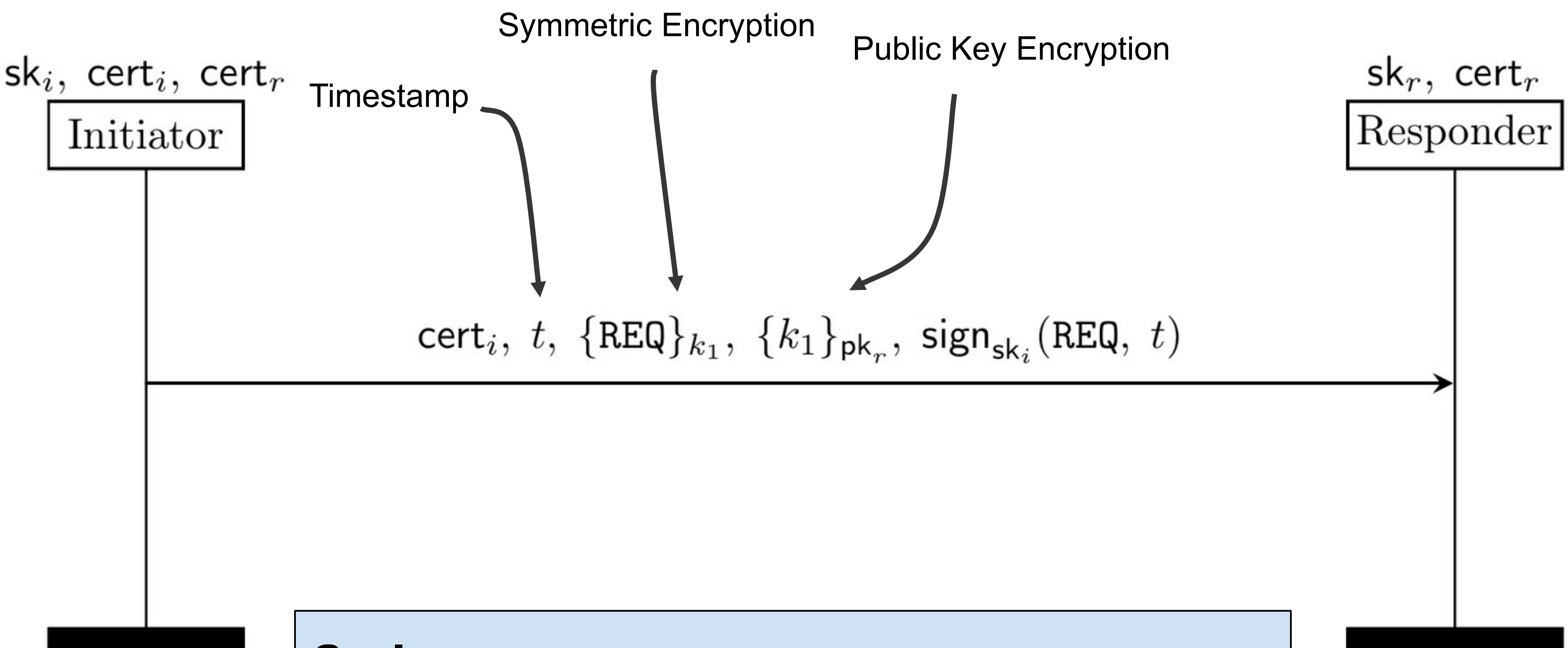
Protocol	Previous verification		New Tamarin analysis [JCCS2019]		
	Year	**Methodology**	**Property**	**Time (s)**	**Attack**
X.509 Mutual Auth	2006	ProVerif	Correlation & Secrecy	5	**NEW ATTACK**
WS Request-Response	2008	F# → ProVerif			
STS-MAC-fix1	2012	Tamarin	Authentication	35	Rediscovered manual attack
STS-MAC-fix2	2012	Tamarin	Authentication	68	Rediscovered manual attack
DRKey & OPT	2014	Coq	Authentication	2640	**NEW ATTACK**
ACME Draft 4	2017	ProVerif	DNS Validation	53	Rediscovered manual attack

WS Security X.509 Mutual Authentication



Goals:
Transmit a request REQ and its response RESP,
Authenticate both parties,
Ensure the response matches the request.

WS Security X.509 Mutual Authentication

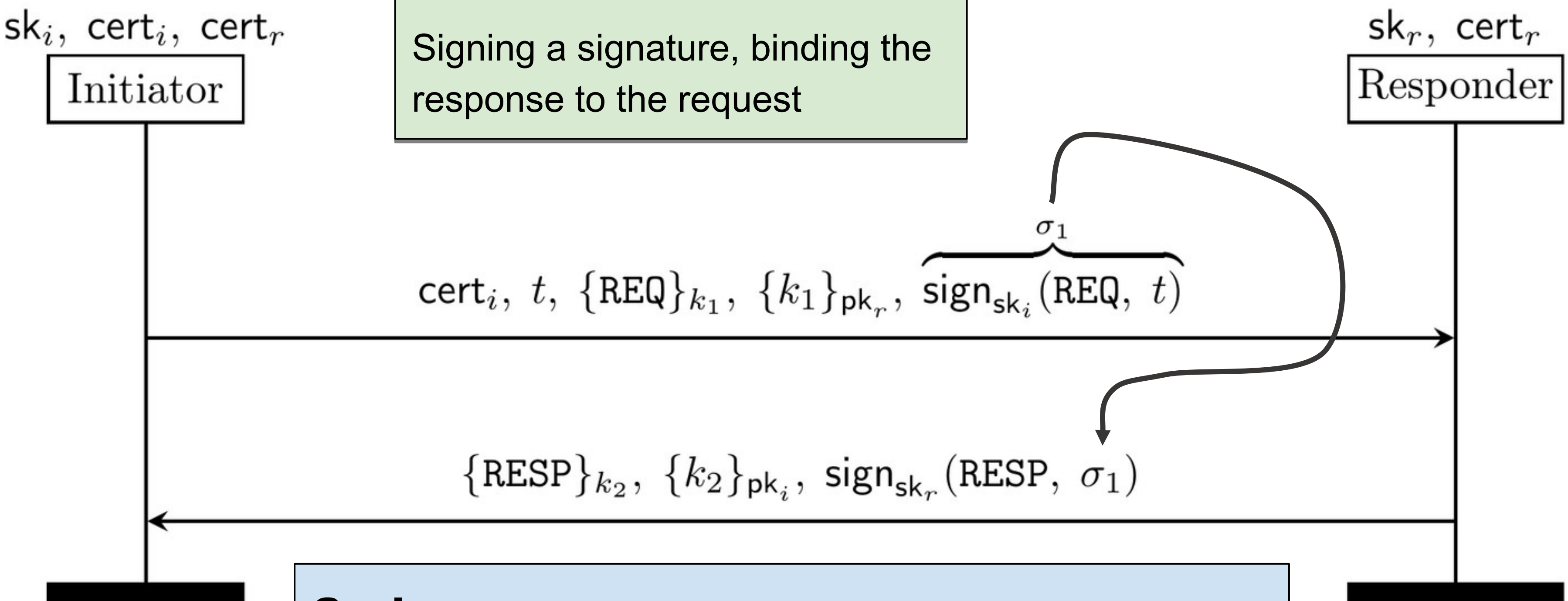


Goals:
Transmit a request REQ and its response RESP,
Authenticate both parties,
Ensure the response matches the request.

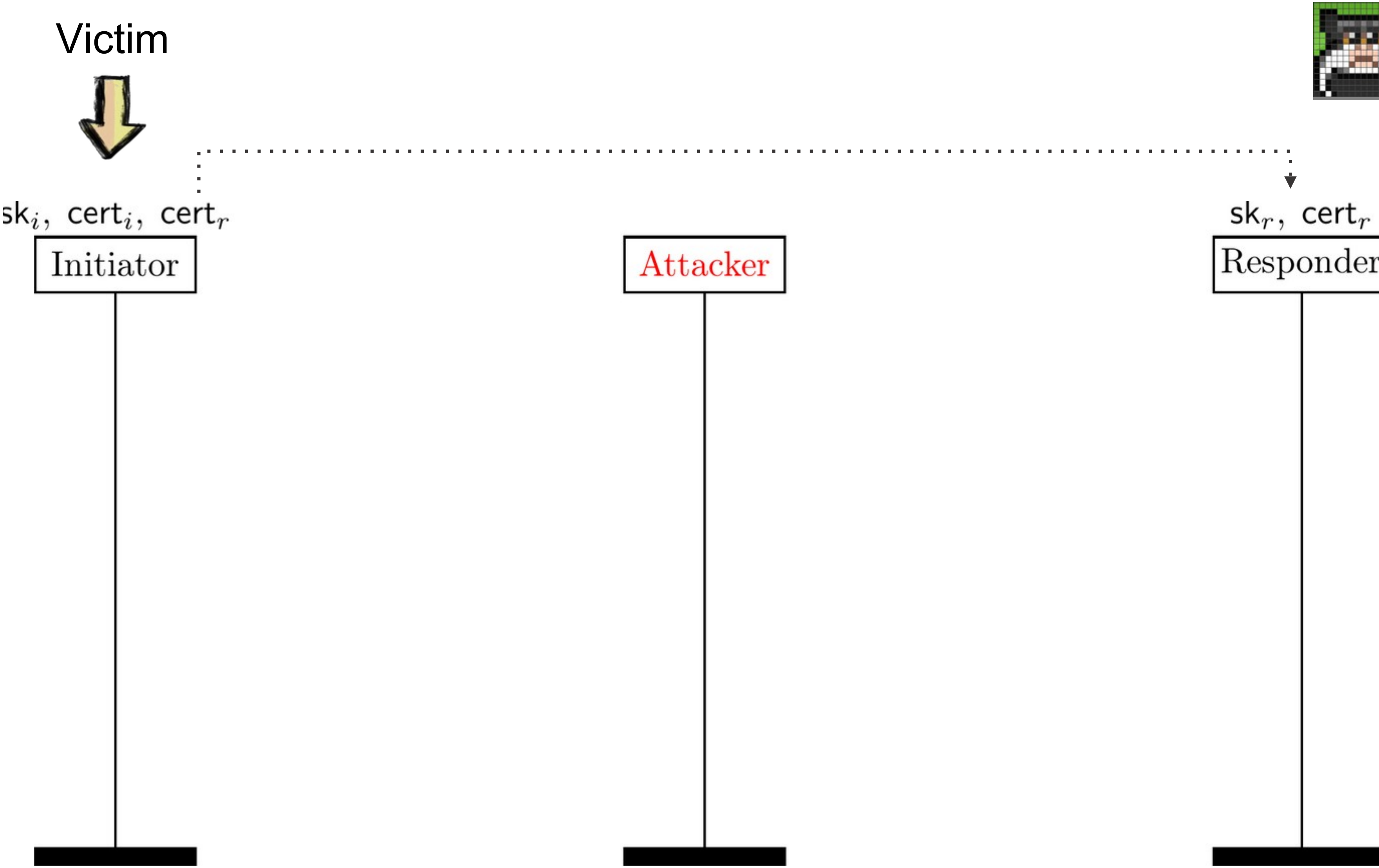
WS Security X.509 Mutual Authentication

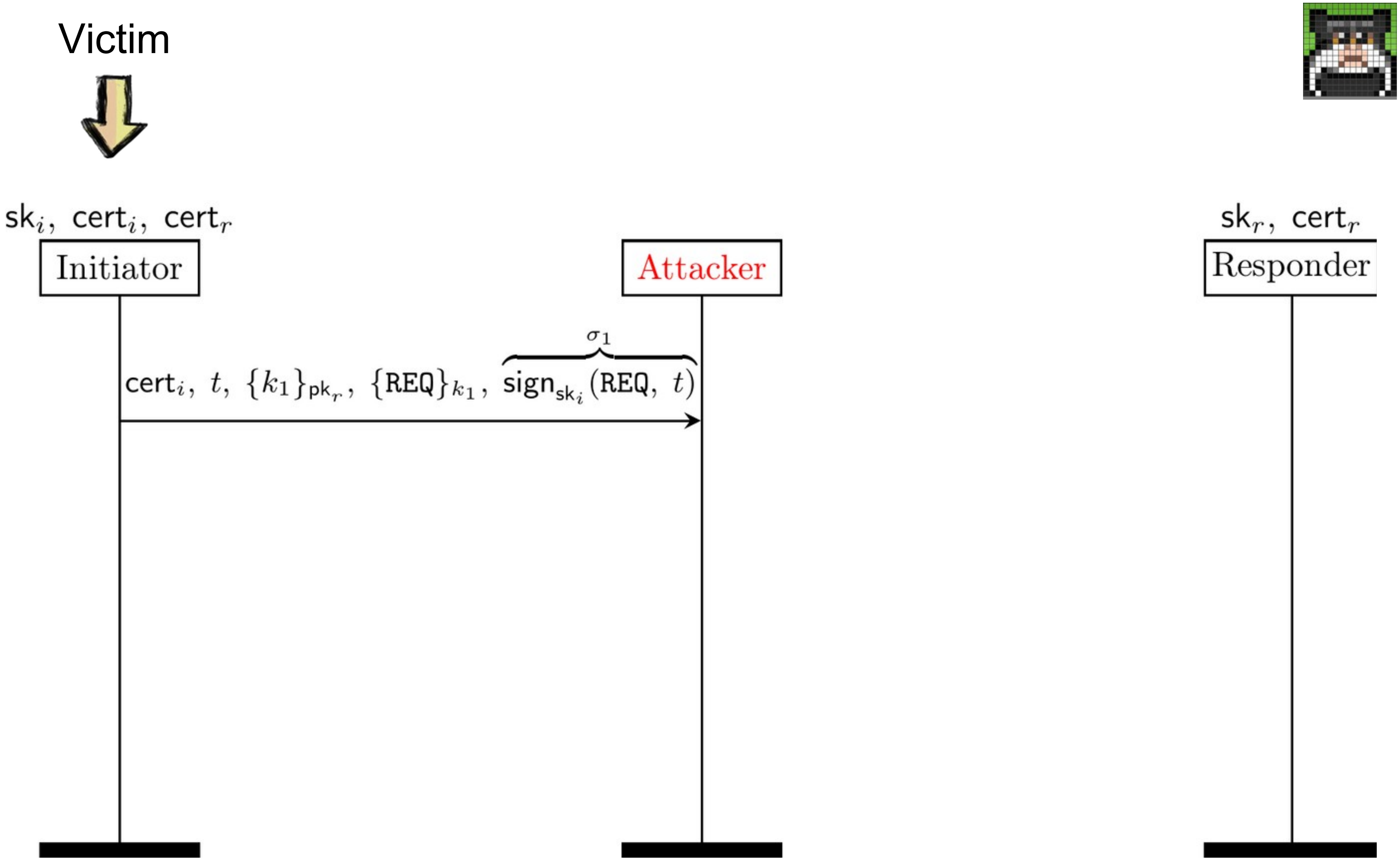
“Signature Confirmation”

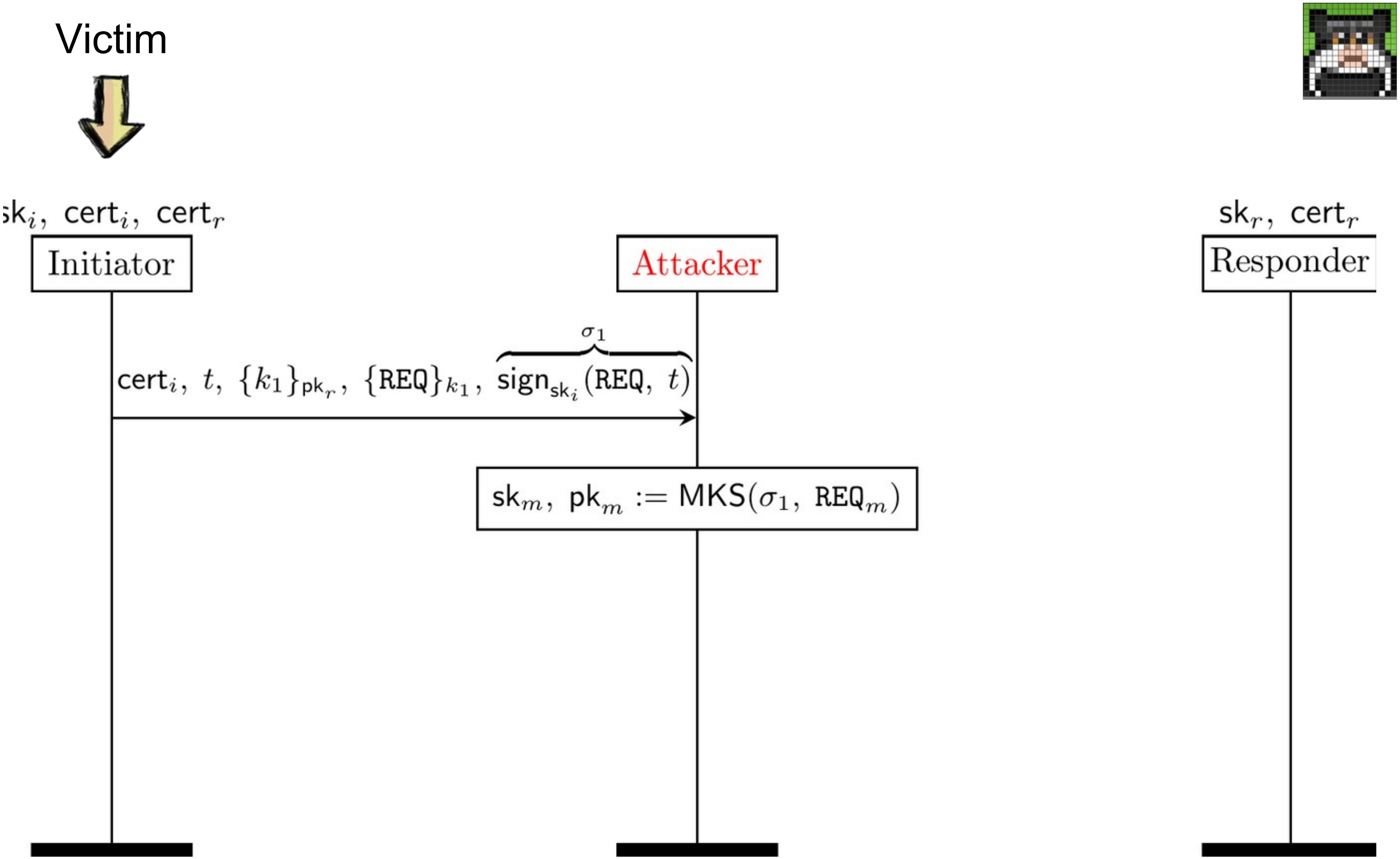
Signing a signature, binding the response to the request

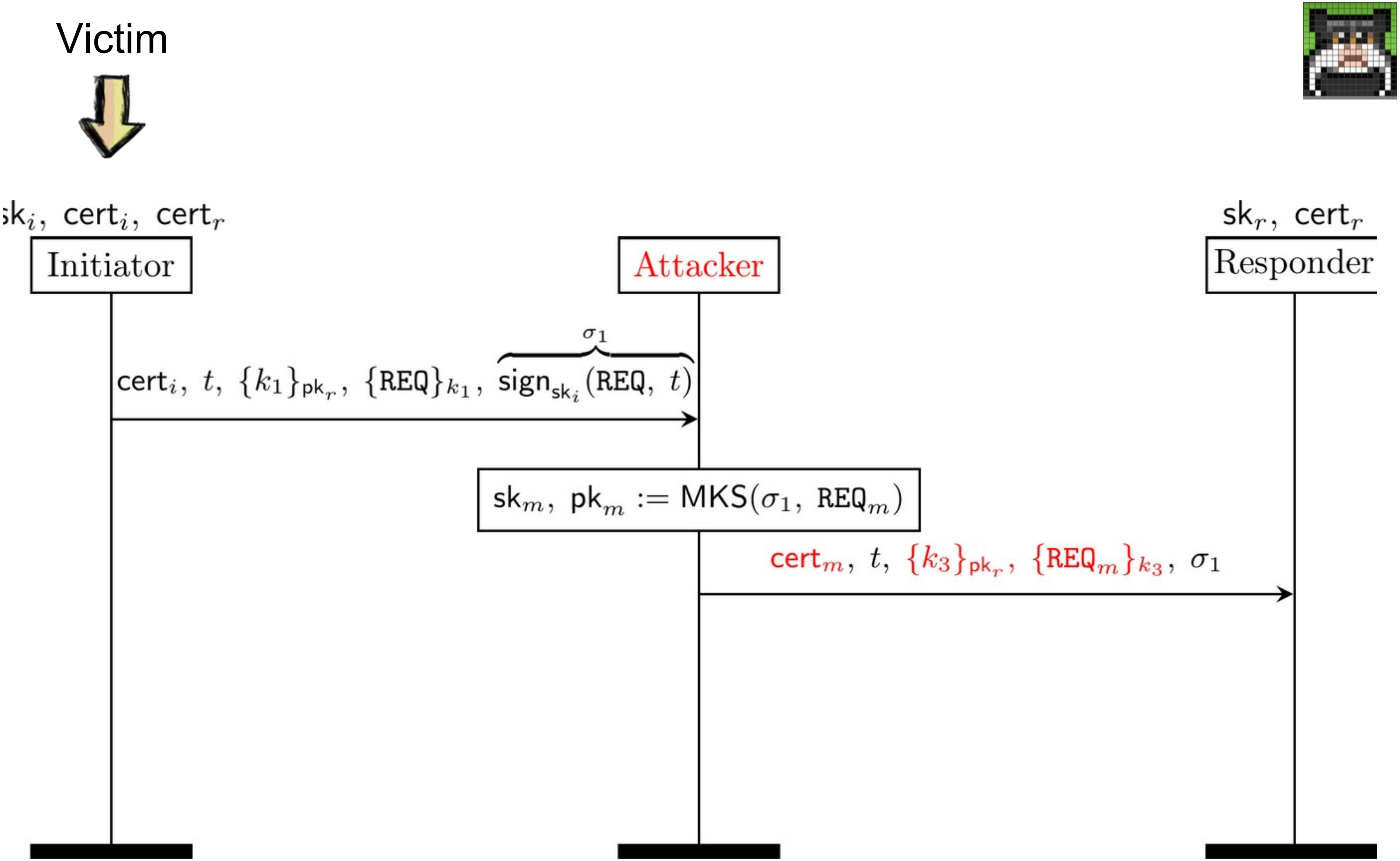


Goals:
Transmit a request REQ and its response RESP,
Authenticate both parties,
Ensure the response matches the request.

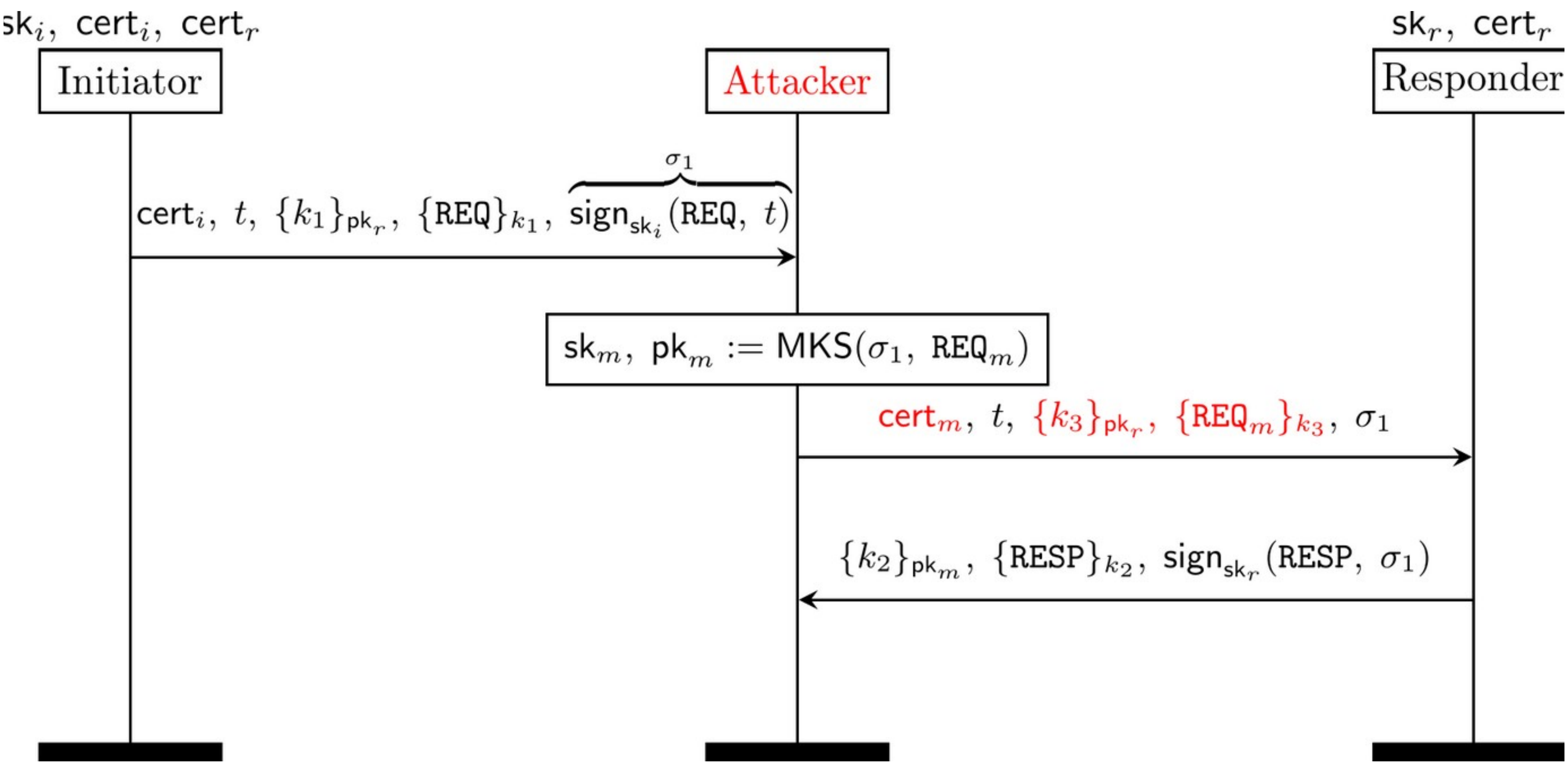

Victim
sk_i, $cert_i$, $cert_r$
Initiator
Attacker
sk_r, $cert_r$
Responder


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


Victim
sk_i, cert_i, cert_r
Initiator
Attacker
sk_r, cert_r
Responder
σ1
cert_i, t, {k1}pk_r, {REQ}k1, sign_sk_i(REQ, t)
sk_m, pk_m := MKS(σ1, REQ_m)

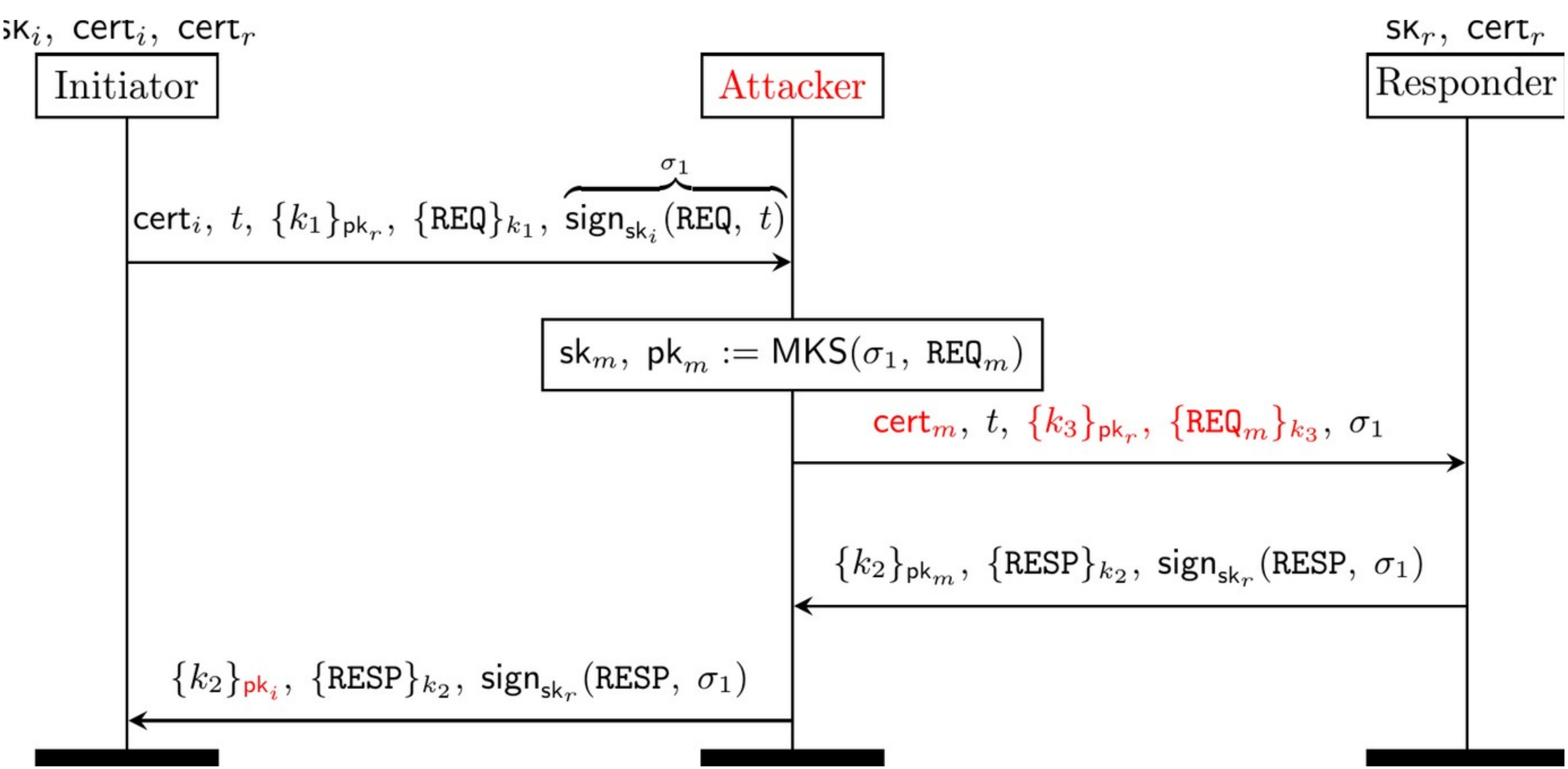

Victim
sk_i, $cert_i$, $cert_r$
Initiator
Attacker
sk_r, $cert_r$
Responder
σ_1
$cert_i$, t, $\{k_1\}_{pk_r}$, $\{REQ\}_{k_1}$, $sign_{sk_i}(REQ, t)$
sk_m, pk_m := $MKS(\sigma_1, REQ_m)$
$cert_m$, t, $\{k_3\}_{pk_r}$, $\{REQ_m\}_{k_3}$, σ_1

Victim

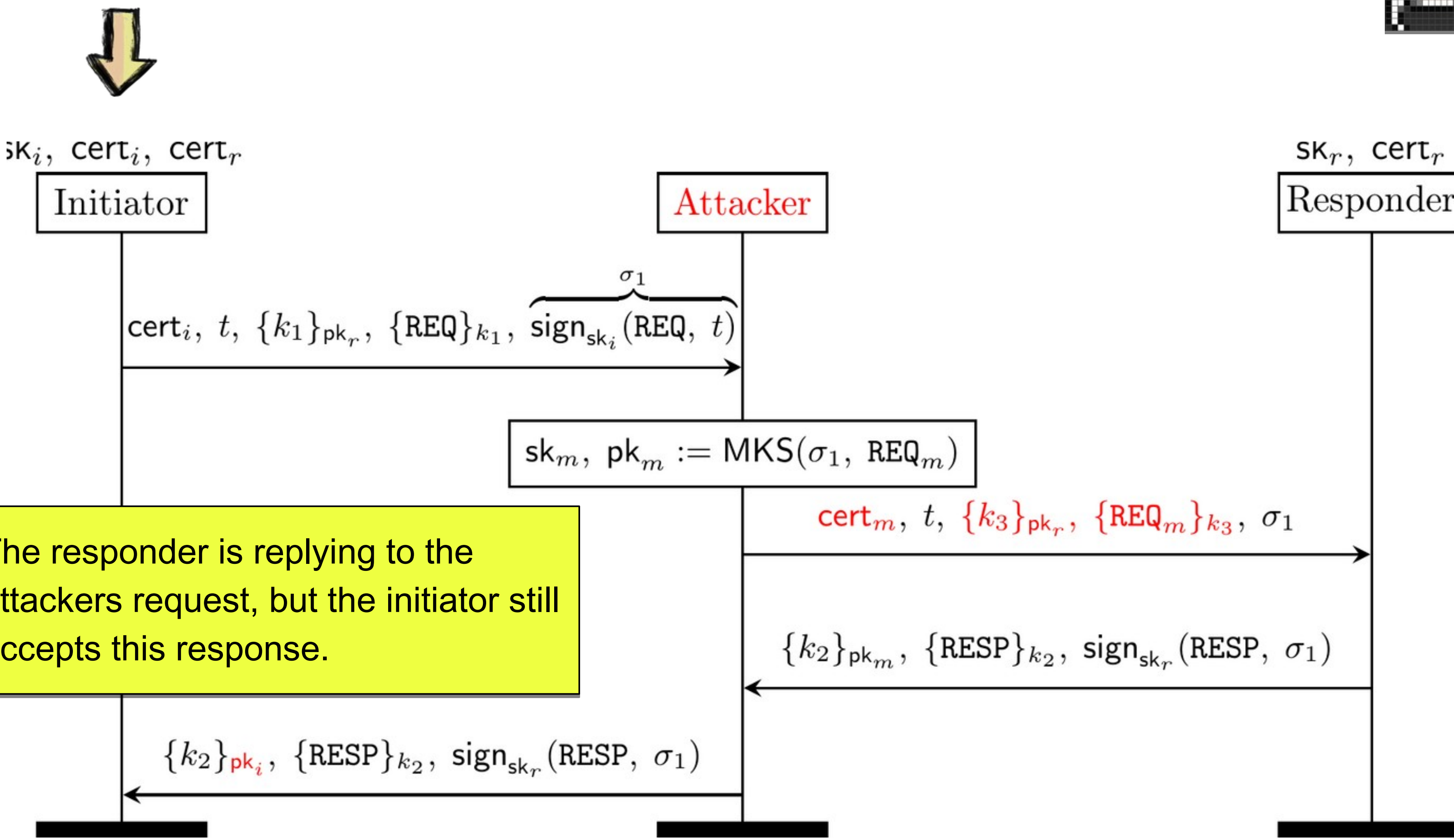


$\mathsf{sk}_i,\ \mathsf{cert}_i,\ \mathsf{cert}_r$
Initiator
Attacker
$\mathsf{sk}_r,\ \mathsf{cert}_r$
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$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

Victim

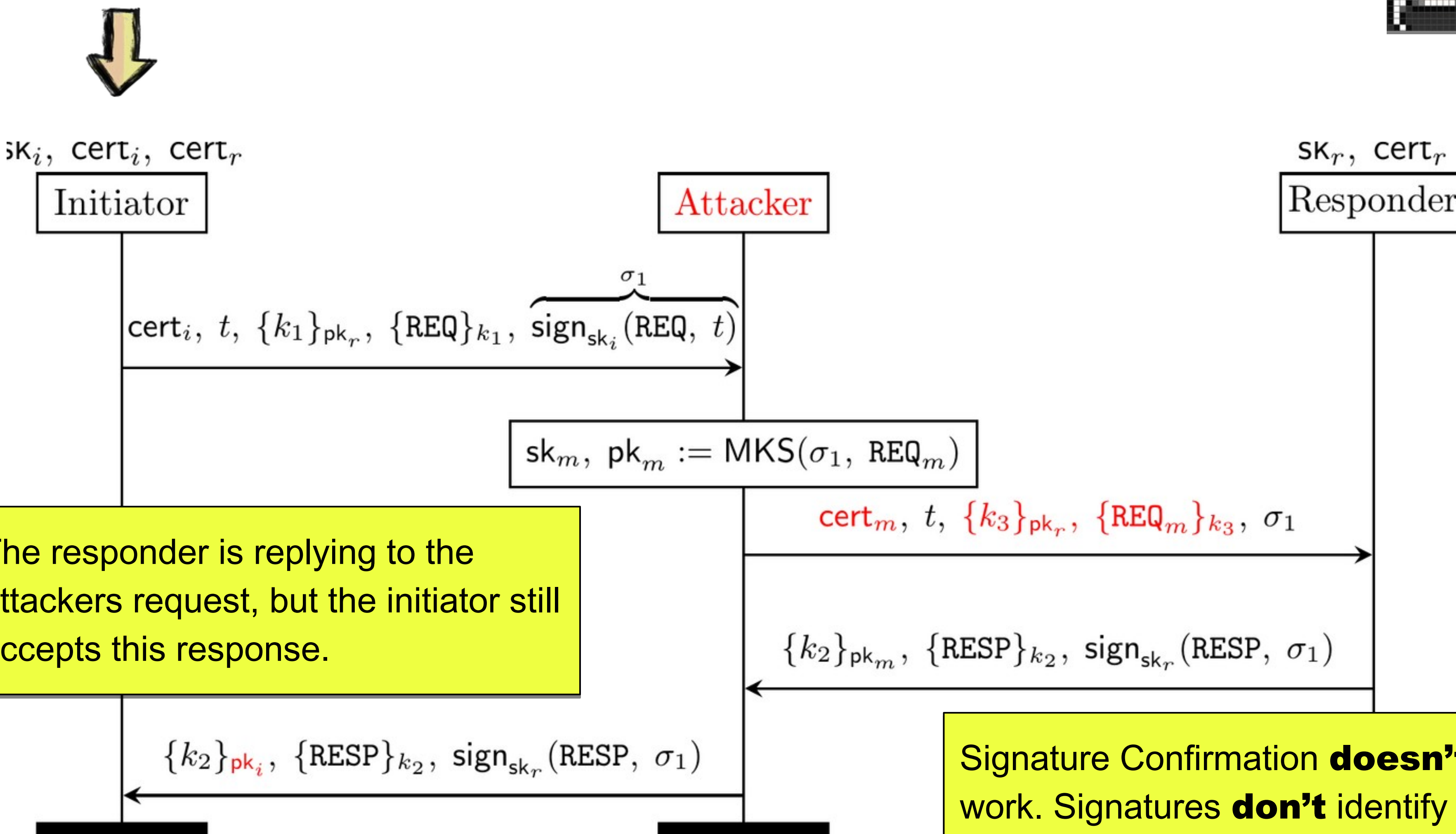

sk_i, cert_i, cert_r
Initiator
Attacker
sk_r, cert_r
Responder
σ_1
$\mathsf{cert}_i,\ t,\ \{k_1\}_{\mathsf{pk}_r},\ \{\mathtt{REQ}\}_{k_1},\ \mathsf{sign}_{\mathsf{sk}_i}(\mathtt{REQ},\ t)$
$\mathsf{sk}_m,\ \mathsf{pk}_m := \mathsf{MKS}(\sigma_1,\ \mathtt{REQ}_m)$
$\mathsf{cert}_m,\ t,\ \{k_3\}_{\mathsf{pk}_r},\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

Victim



The responder is replying to the attackers request, but the initiator still accepts this response.

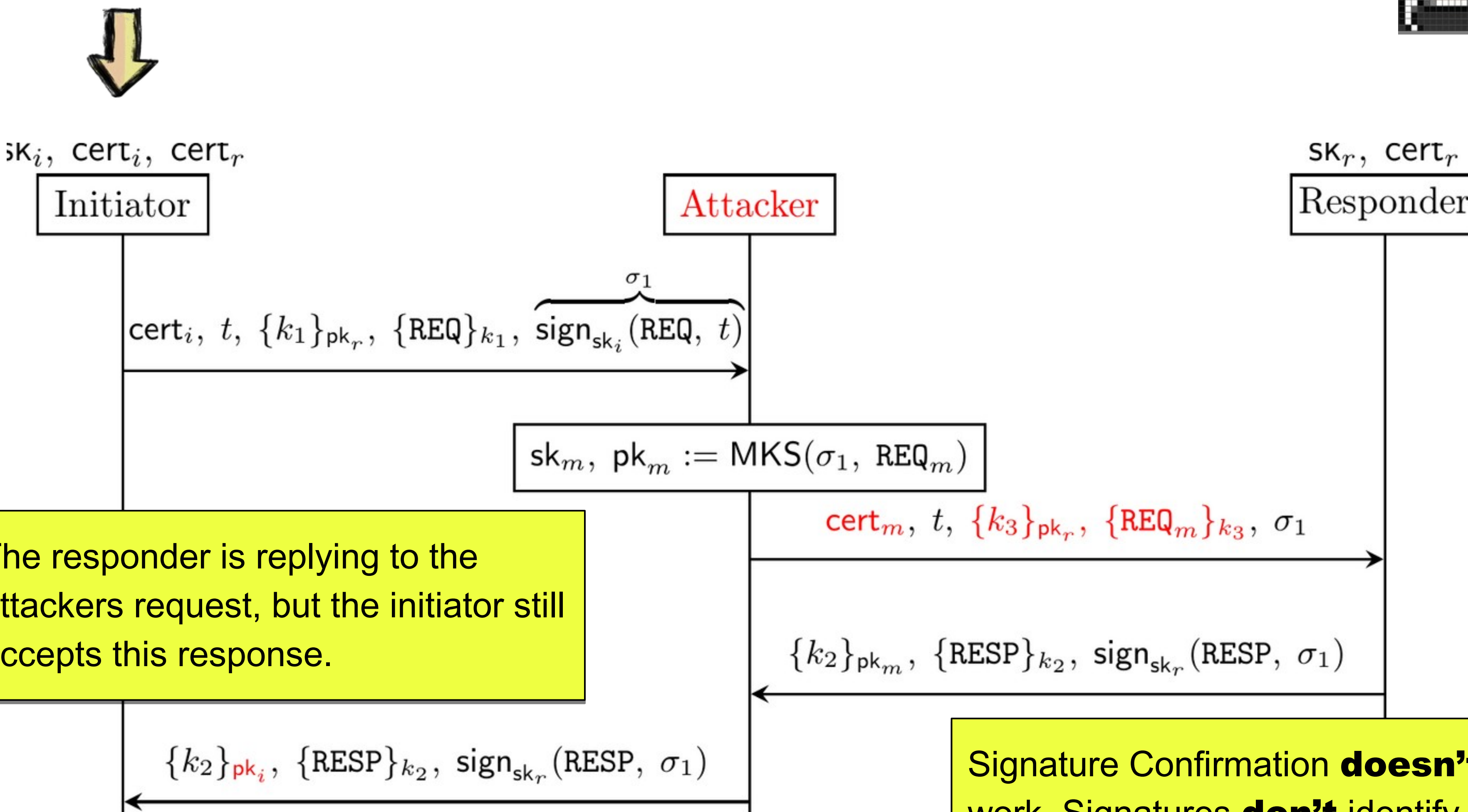
Victim



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.

Victim



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.

Attacker violates the guarantees of the Initiator:
- Can modify REQ to REQ_m (or leave unchanged)
- Learns k_2 and content of RESP (but not REQ)
- There is no Responder that thinks they are talking to this Initiator

Other primitives example: Diffie-Hellman

Diffie-Hellman

Investigation:

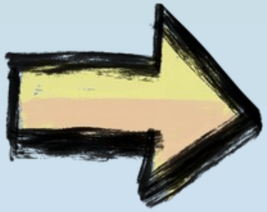
- Prime order groups / curves are encoded in various complex ways
- Lead to subtly different classes of behaviours
  - Prime order groups (= traditional DY model)
  - "Nearly-prime" order groups (small cogroup)
  - Composite groups
  - Single coordinate ladders (for EC)
  - General invalid curve points (for EC)

Diffie-Hellman

Investigation:

- Prime order groups / curves are encoded in various complex ways
- Lead to subtly different classes of behaviours
    - Prime order groups (= traditional DY model)
    - "Nearly-prime" order groups (small cogroup)
    - Composite groups
    - Single coordinate ladders (for EC)
    - General invalid curve points (for EC)

We give symbolic models for each, and for the implemented "checks"



Tamarin finds new attacks automatically
Go's standard crypto library will get new API
Better checks in Cloudflare's standard libraries

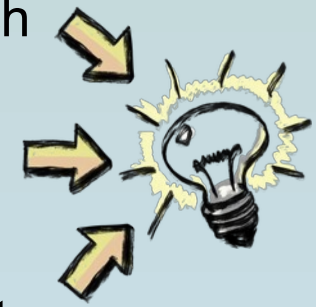
Stepping back

Sometimes ideas escalate!

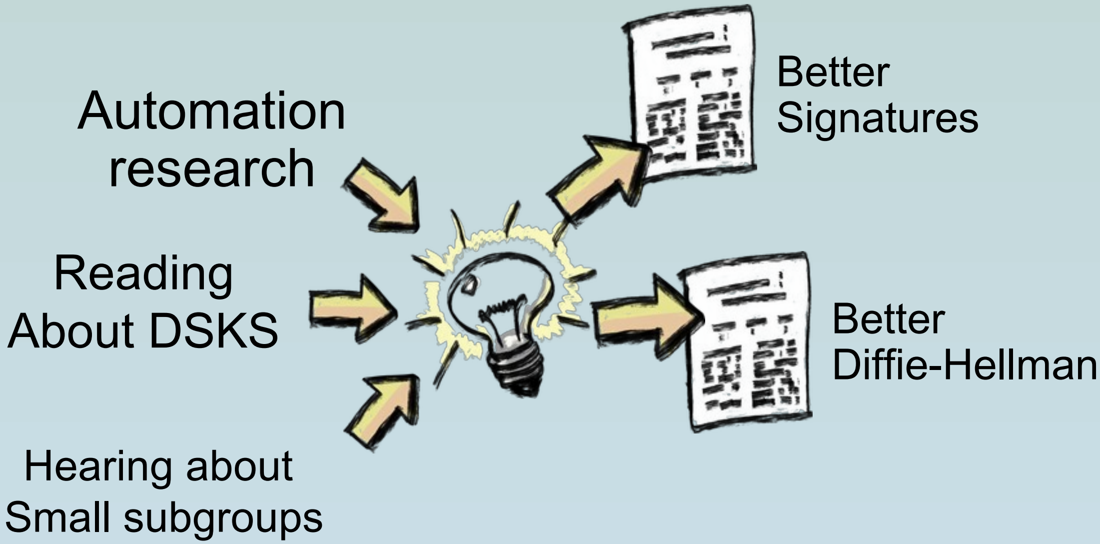
Automation
research

Reading
About DSKS

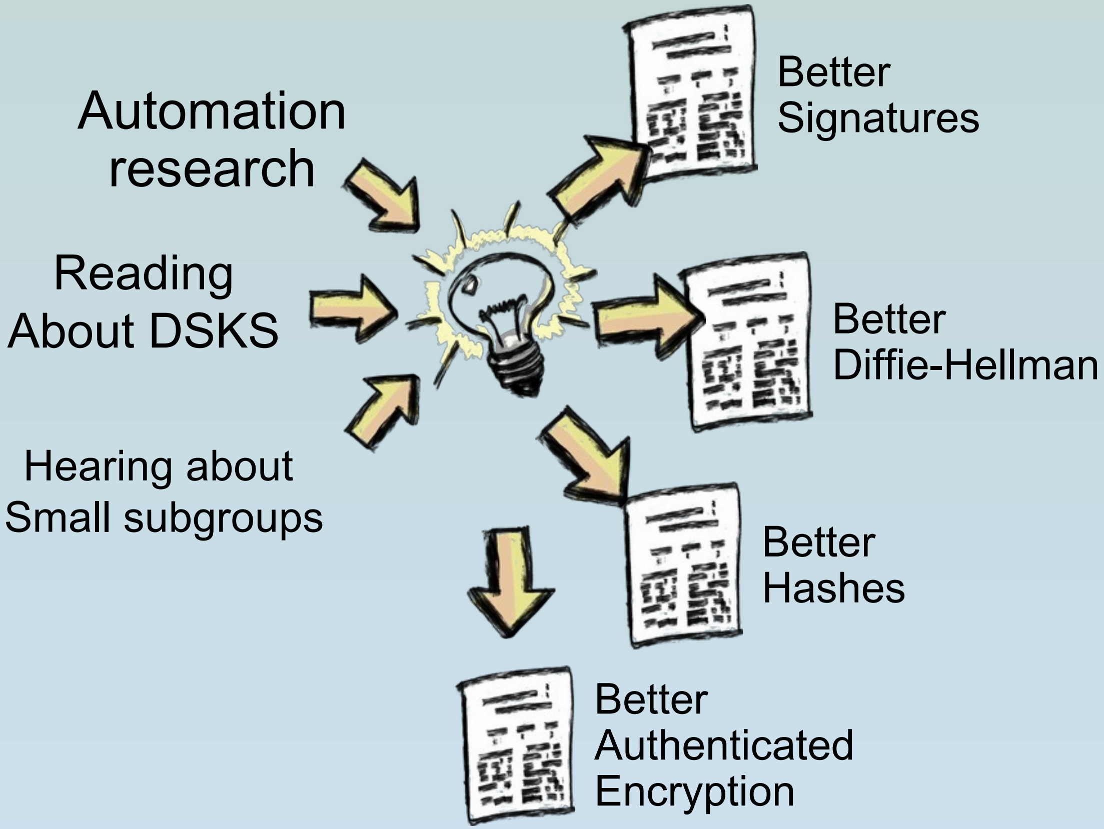
Hearing about
Small subgroups



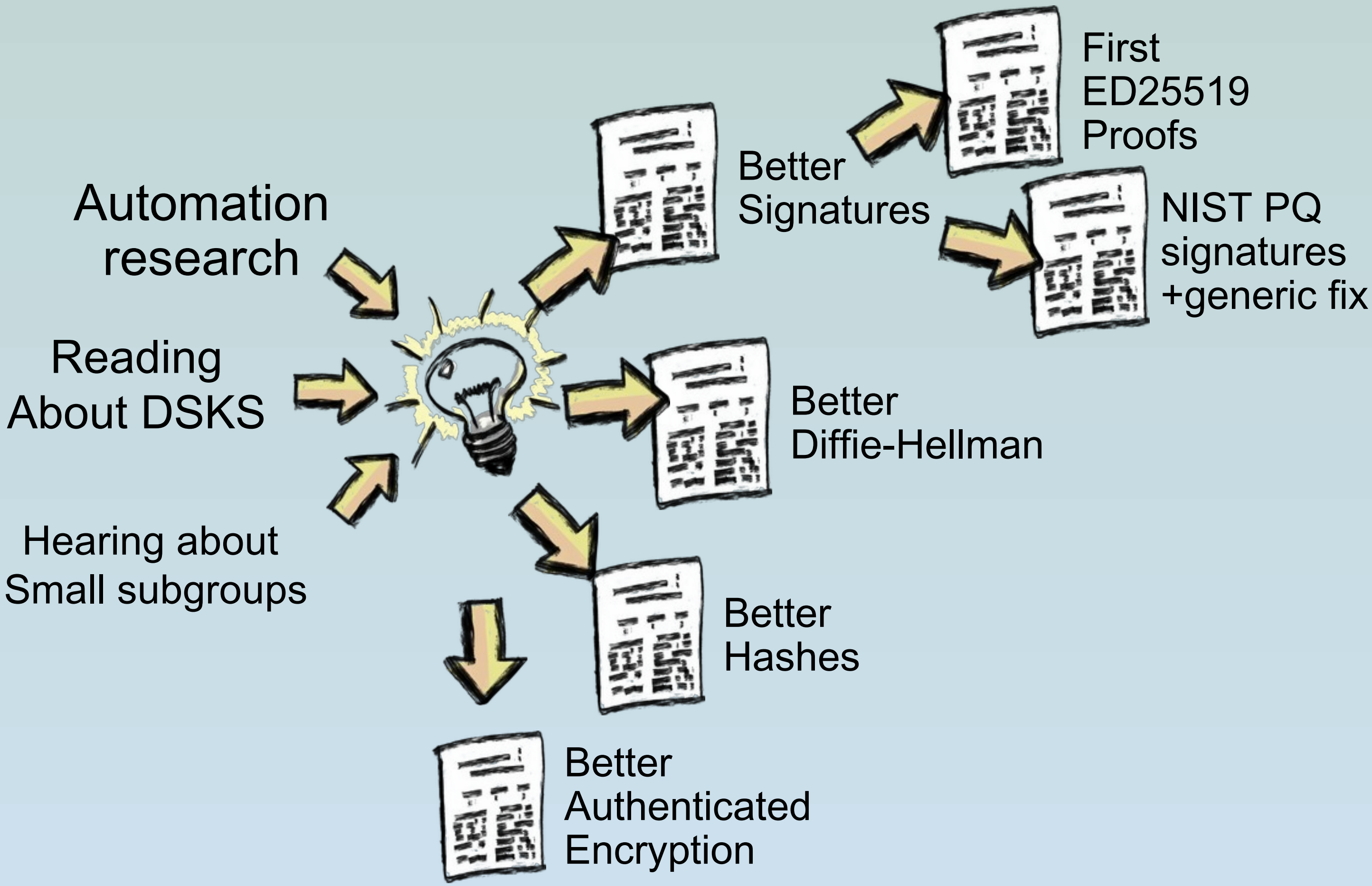
Sometimes ideas escalate!



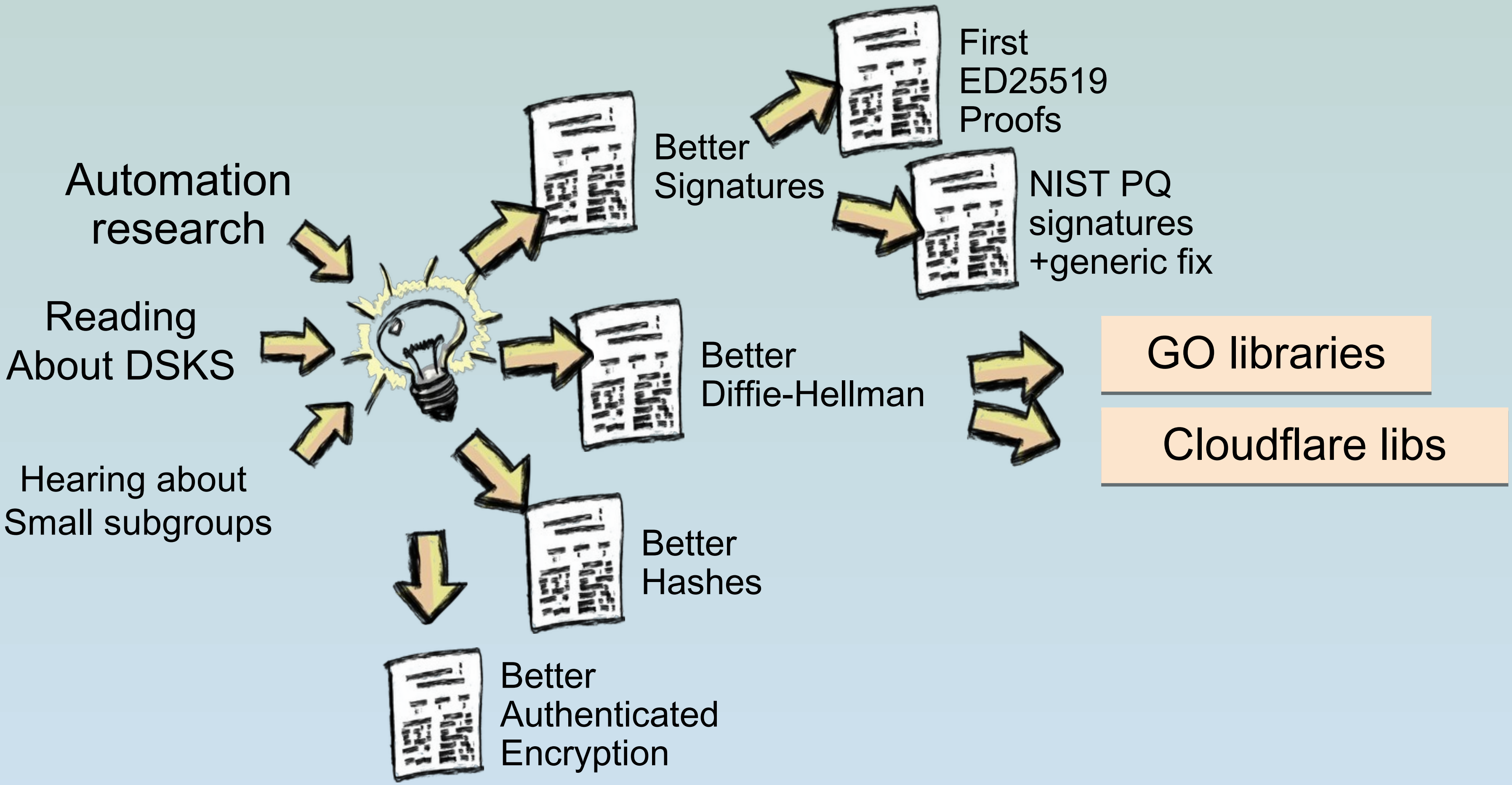
Sometimes ideas escalate!



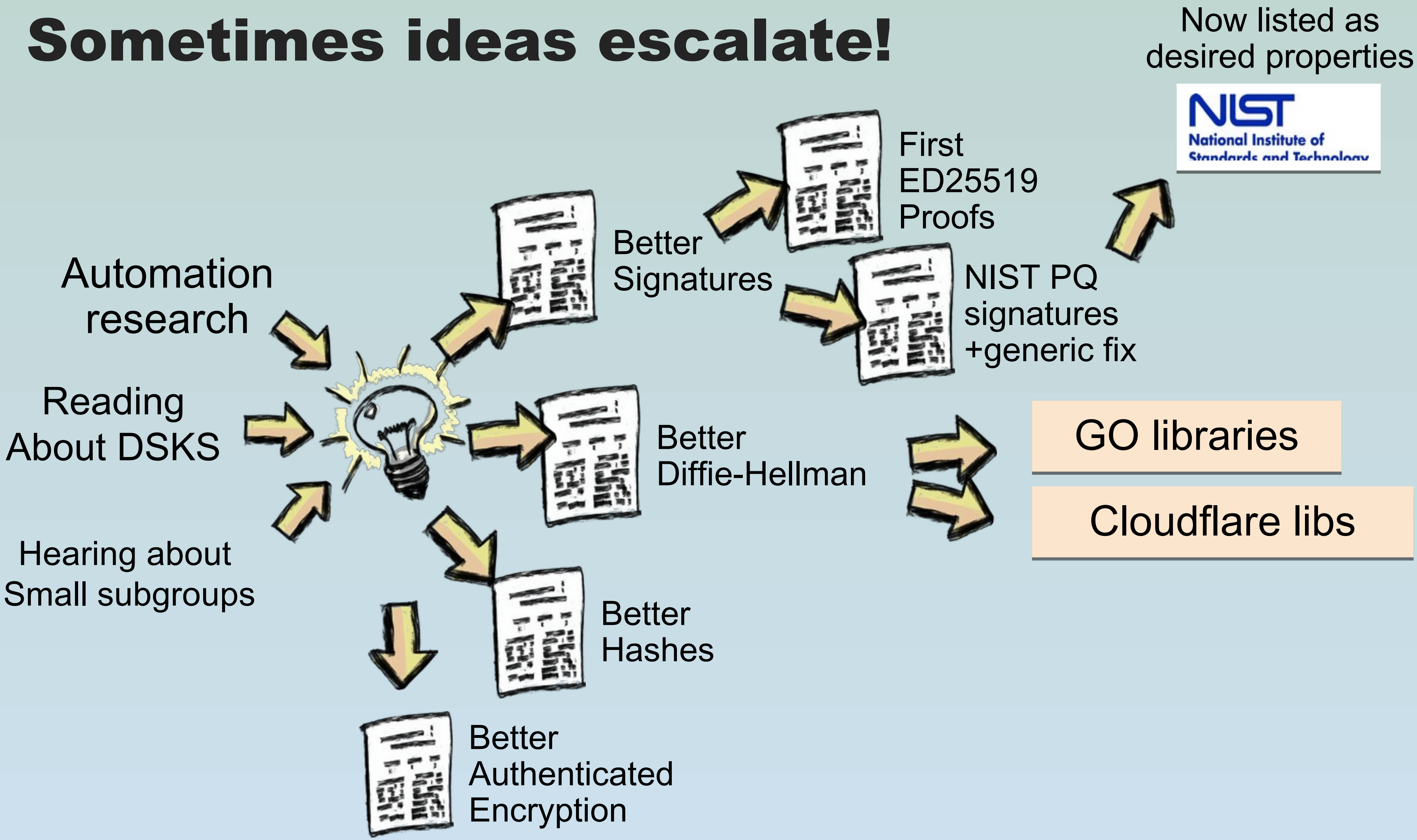
Sometimes ideas escalate!



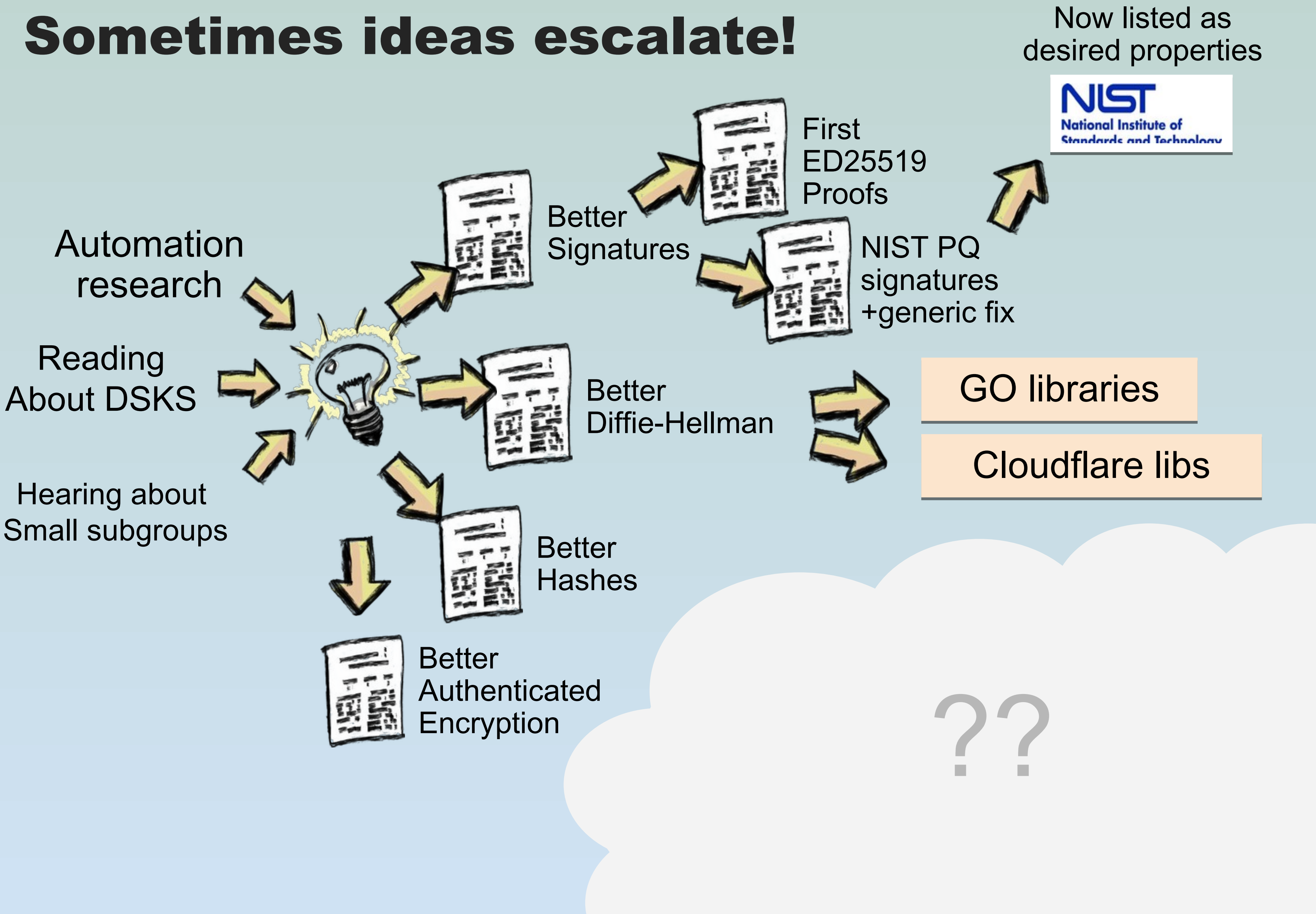
Sometimes ideas escalate!



Sometimes ideas escalate!



Sometimes ideas escalate!



Wider question for future developments:

Which attacks are covered by computational protocol proofs, but cannot be captured symbolically?

My original intuition:
Probably there are plenty of examples.

My current intuition:
Not so sure anymore there are many interesting ones!

What about larger protocols?

- Case studies can run large too
- Recent example: SPDM 1.2
  - 75 rules
  - 2500 lines of code
  - 40 lemmas
  - First CVE automatically found by Tamarin
    - Individual modes secure
    - Composition completely breaks mutual authentication for one mode!
- Back to David!

Backup slides

Computational protocol proofs would capture this, right?

In general, **no**

How it works instead in most protocol proofs:

- Setup phase (honestly) generates key pairs for every party
- Adversary can correct some of these parties to learn private keys
- The analysis only considers public keys from the setup phase

Consequence:

- The proof gives no guarantees for maliciously generated keys

Definition 4.2 (Multi-Stage security). *Let* KE *be a multi-stage key exchange protocol with prop-*
erties (M, AUTH, FS, USE, REP
with KE *via the queries define*

- Corrupt(U) or Corrupt(U, V, pssid): The first query is only used in the public-key (pMSKE) variant, the second query only in the pre-shared secret (sMSKE) variant. Provide the adversary with the corresponding long-term secret, i.e., sk_U (pMSKE), resp. $\mathsf{pss}_{U,V}(\mathsf{pssid})$ (sMSKE).

Setup. *The challenger chooses the test bit* $b_{\mathsf{test}} \leftarrow_\$ \{0,1\}$ *at random and sets* lost ← false. *In the public-key variant (pMSKE), it furthermore generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$.

Modern signature schemes

At least modern schemes like **Ed25519** satisfy these properties?

- **Ed25519-Original:**
  - Provides only existential unforgeability
  - Does not provide guarantees for maliciously generated public keys (as documented)
- **Ed25519-IETF:**
  - Provides some guarantees, notably strong unforgeability, but not all
- **Ed25519-LibSodium:**
  - Provides the strongest guarantees including wrt malicious keys

Oh, there is also the NIST Competition for post-quantum secure signature schemes. Surely they are fine, freshly designed!

[BCJZ2021] IEEE S&P 2021: *Ed25519 Signature Schemes: Theory and Practice*

NIST Post-Quantum Signature competition

To our surprise, previous NIST competition rounds only require existential unforgeability

	Round 3 scheme		malicious strong univ. exclusive ownership	message-bound signatures	no re-signing without message	Conclusion
main	CRYSTALS-Dilithium		✓	✓	✓	✓
	FALCON		✗	✓	✗	✗
	Rainbow	Standard	✗	✓	✗	✗
	Rainbow	CZ & Compr.	•	✓	✗	✗
alternate	GeMSS		✗	✗	✗	✗
	Picnic		✓	✓	✓	✓
	SPHINCS$^+$		•	✓	•	•

We show a generic BUFF transform to provably achieve all these properties

[CDFFJ2021] IEEE S&P 2021: *BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures*

Automated Certificate Issuance, deployed in 2015

Over 1 million certificates issued every day!

Idea

- Proof of Domain Ownership
- Challenge Response Protocol
- Prove you control the DNS Records for a website

April-May 2015

ISRG Engages NCC Group for Let's Encrypt Audit

[Acme] Signature misuse vulnerability in draft-barnes-acme-04

- *From*: Andrew Ayer <agwa at andrewayer.name>
- *To*: acme at ietf.org
- *Date*: Tue, 11 Aug 2015 08:52:05 -0700
- *List-id*: Automated Certificate Management Environment <acme.ietf.org>

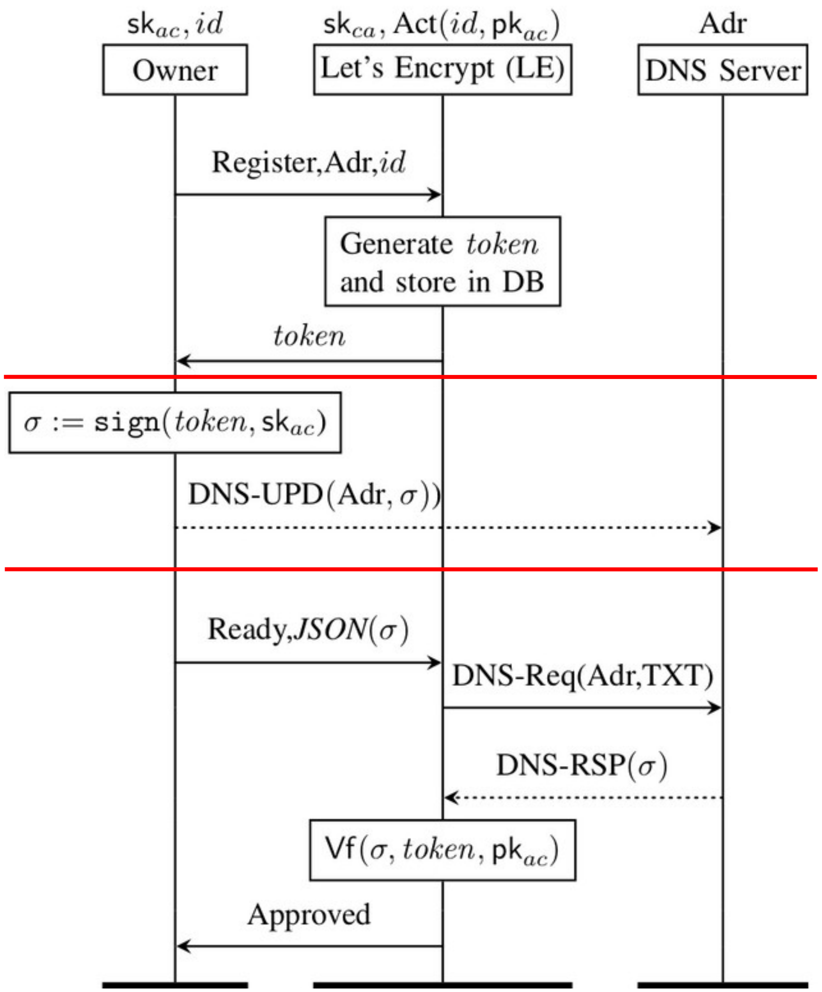
```
I recently reviewed draft-barnes-acme-04 and found vulnerabilities in
the DNS, DVSNI, and Simple HTTP challenges that would allow an attacker
to fraudulently complete these challenges.
```

11th August

15th September

First Let's Encrypt Free Certificate Goes Live

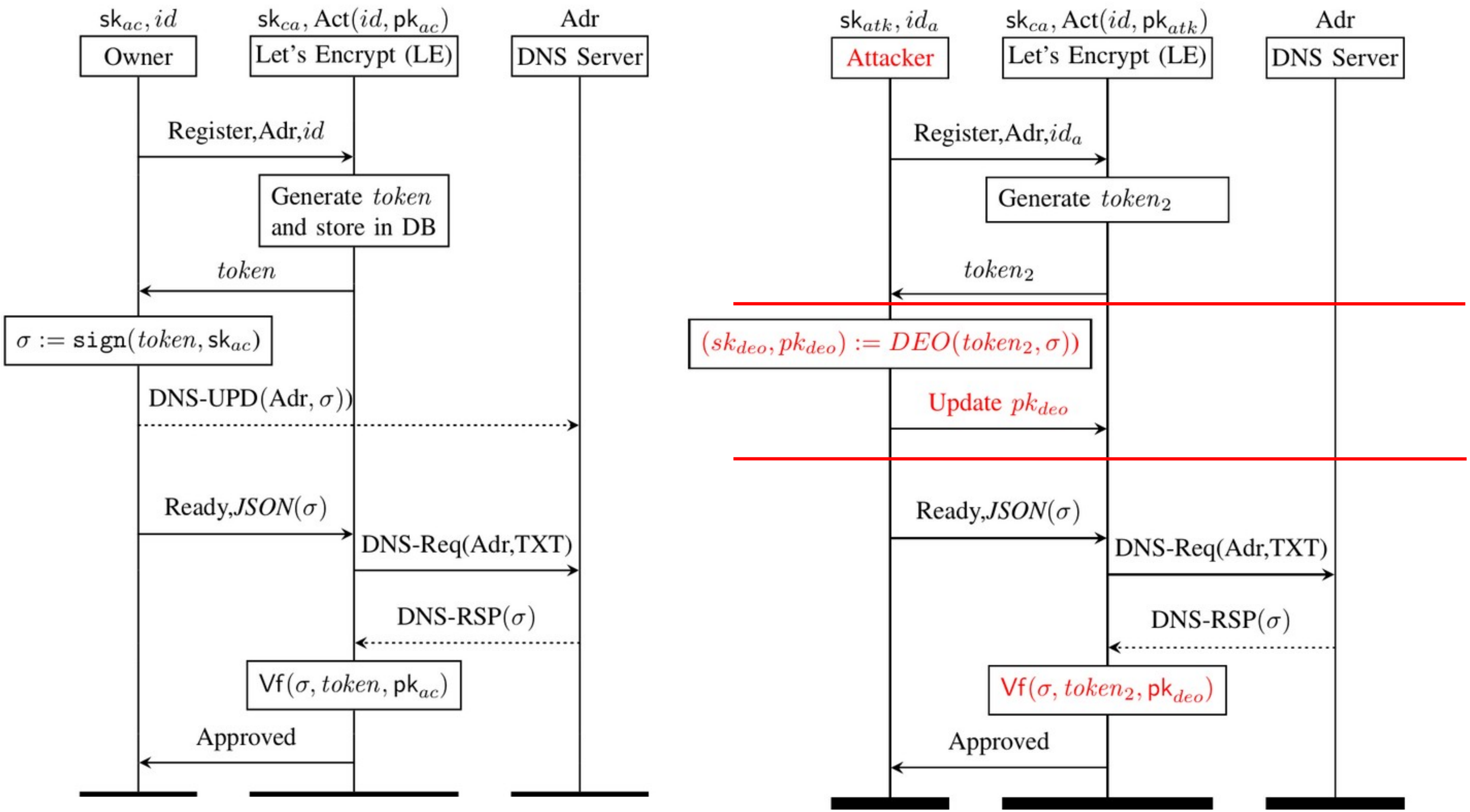
Let's Encrypt hit a major milestone today when its first free and automated cert went live



Challenge

Response

Verify


sk_{ac}, id
Owner
sk_{ca}, Act(id, pk_{ac})
Let's Encrypt (LE)
Adr
DNS Server
Register,Adr,id
Generate $token$ and store in DB
$token$
$\sigma := \mathtt{sign}(token, \mathsf{sk}_{ac})$
DNS-UPD(Adr, σ))
Ready,$JSON(\sigma)$
DNS-Req(Adr,TXT)
DNS-RSP(σ)
$\mathsf{Vf}(\sigma, token, \mathsf{pk}_{ac})$
Approved
sk_{atk}, id_a
Attacker
sk_{ca}, Act(id, pk_{atk})
Let's Encrypt (LE)
Adr
DNS Server
Register,Adr,id_a
Generate $token_2$
$token_2$
$(sk_{deo}, pk_{deo}) := DEO(token_2, \sigma))$
Update pk_{deo}
Ready,$JSON(\sigma)$
DNS-Req(Adr,TXT)
DNS-RSP(σ)
$\mathsf{Vf}(\sigma, token_2, \mathsf{pk}_{deo})$
Approved

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 that passes **Verify**,
   but the adversary never submitted the message in step 2.

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

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]

Note:
Definition says nothing about what should hold for maliciously generated keys

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