IS&C

UNIVERSITÄT DES SAARLANDES

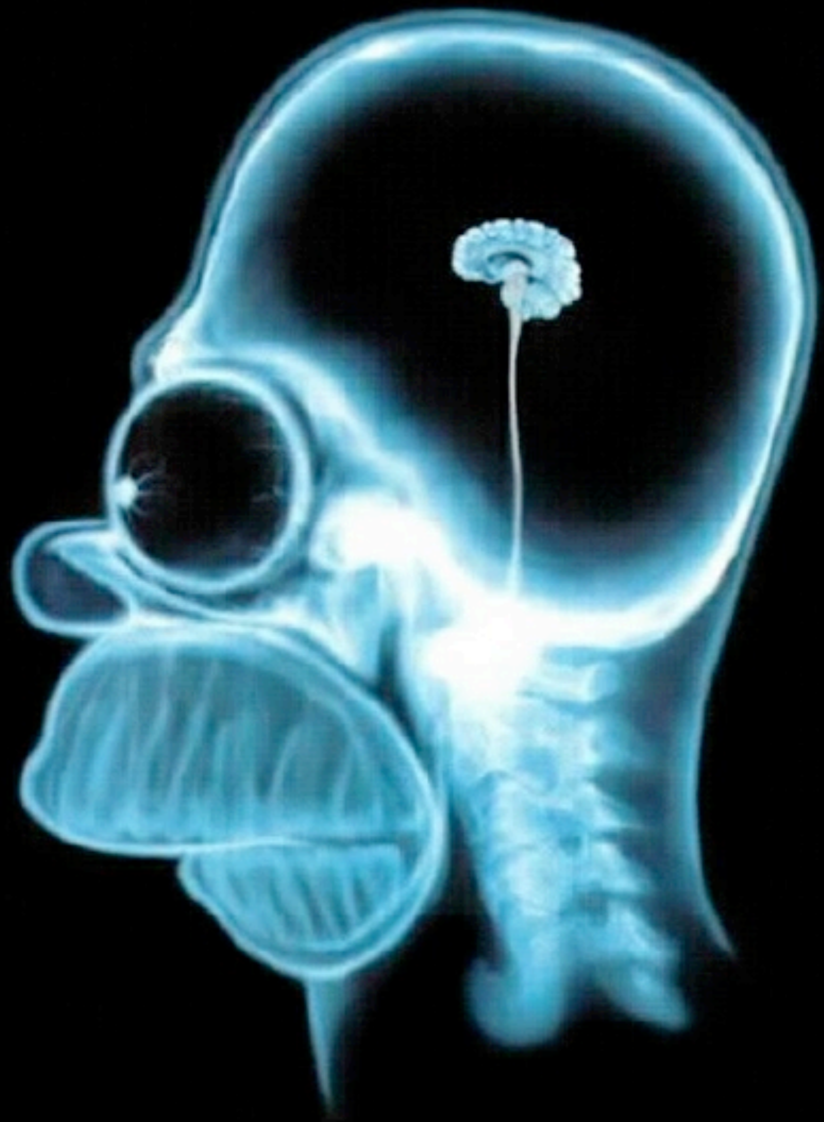
Type-checking Zero-knowledge

Cătălin Hrițcu

Saarland University, Saarbrücken, Germany

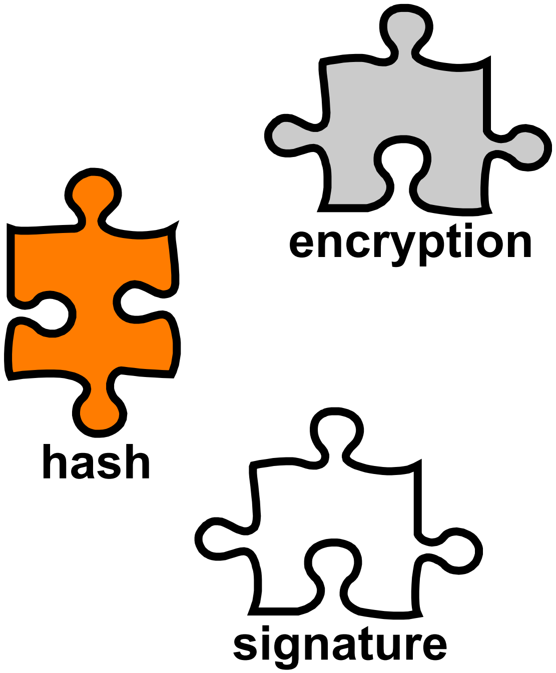
Joint work with: Michael Backes and Matteo Maffei

Zero-knowledge



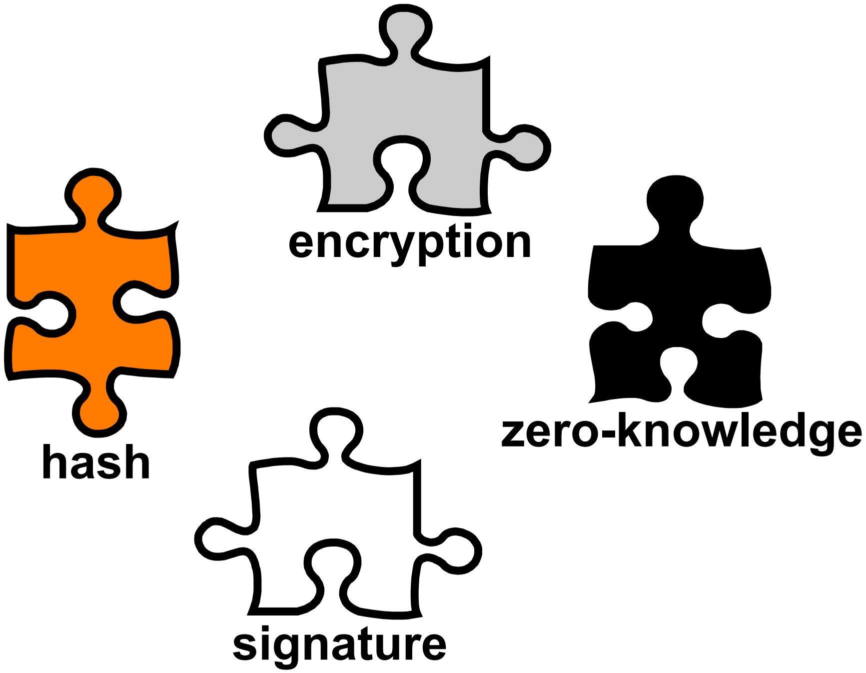
Security Protocols

designer’s toolbox



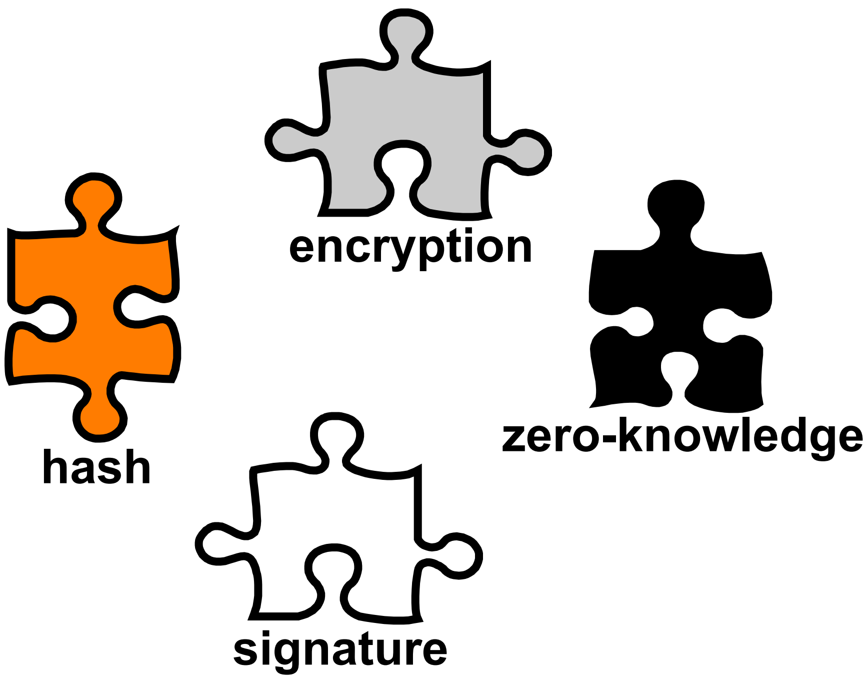
Security Protocols

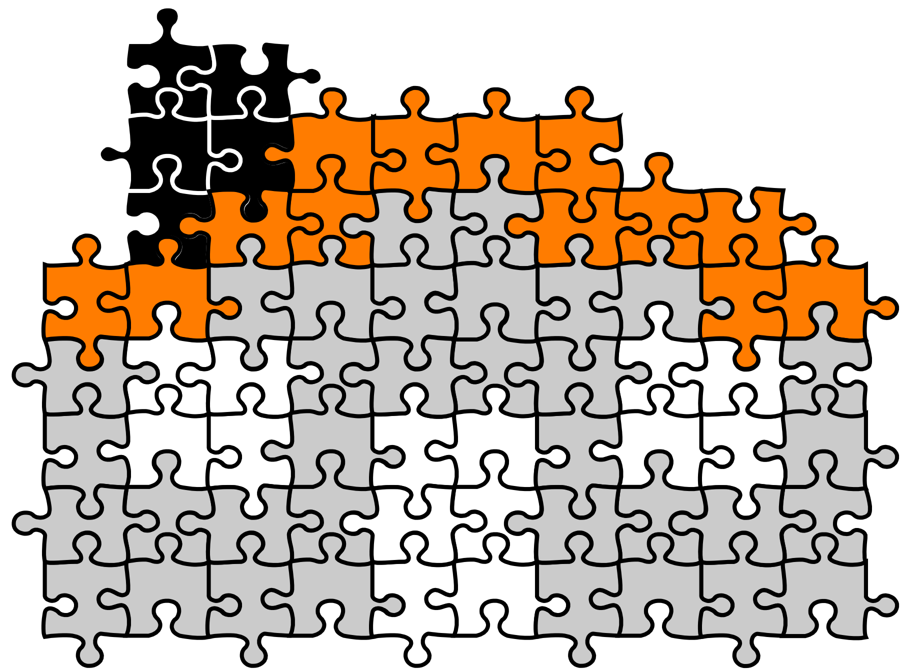
designer’s toolbox



Security Protocols

designer's toolbox





Zero-knowledge Proofs

- Powerful primitives
  - Prove the validity of a statement without revealing anything else
    - For instance, prove to know an object with certain properties without revealing this witness

Zero-knowledge Proofs

- Powerful primitives
  - Prove the validity of a statement without revealing anything else
    - For instance, prove to know an object with certain properties without revealing this witness
- Early constructions general, but terribly inefficient
  - Very limited practical impact

Zero-knowledge Proofs

- Powerful primitives
  - Prove the validity of a statement without revealing anything else
    - For instance, prove to know an object with certain properties without revealing this witness
- Early constructions general, but terribly inefficient
  - Very limited practical impact
- More recent research provided
  - Efficient constructions for special classes of statements
  - Constructions for non-interactive zero-knowledge

Applications that use ZK

unique security features of ZK allows designing protocols fulfilling seemingly *conflicting requirements*

e-voting (Civitas)

“trusted” computing (DAA)

p2p (PseudoTrust)

privacy

+

verifiability

→

e-voting (Civitas)

Applications that use ZK

unique security features of ZK allows designing protocols fulfilling seemingly *conflicting requirements*

"trusted" computing (DAA)

p2p (PseudoTrust)

privacy

+

verifiability

→

e-voting (Civitas)

anonymity

+

remote attestation

→

“trusted” computing (DAA)

p2p (PseudoTrust)

privacy

+

verifiability

→

e-voting (Civitas)

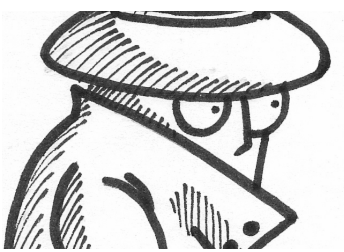
anonymity

+

remote attestation

→

“trusted” computing (DAA)



pseudonymity

+

trust

→

p2p (PseudoTrust)

Verifying Protocols Using ZK

- No verification tool for protocols using ZK

Verifying Protocols Using ZK

- No verification tool for protocols using ZK
- Security protocols are hard to get right

Verifying Protocols Using ZK

- No verification tool for protocols using ZK
- Security protocols are hard to get right
- Automated verification can prevent errors

Verifying Protocols Using ZK

- No verification tool for protocols using ZK
- Security protocols are hard to get right
- Automated verification can prevent errors
- Our goal
  - ***To automatically analyze protocols using ZK in an efficient and scalable way***

Verifying Protocols Using ZK

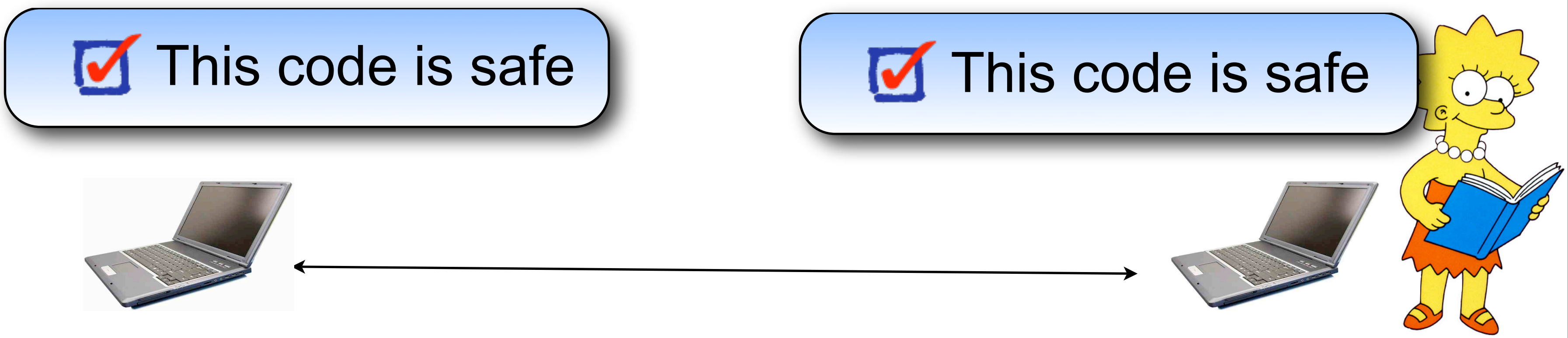
- No verification tool for protocols using ZK
- Security protocols are hard to get right
- Automated verification can prevent errors
- Our goal
  - ***To automatically analyze protocols using ZK in an efficient and scalable way***
- We built first ***type system for zero-knowledge***

Type System

- Type-checking fully automated and efficient

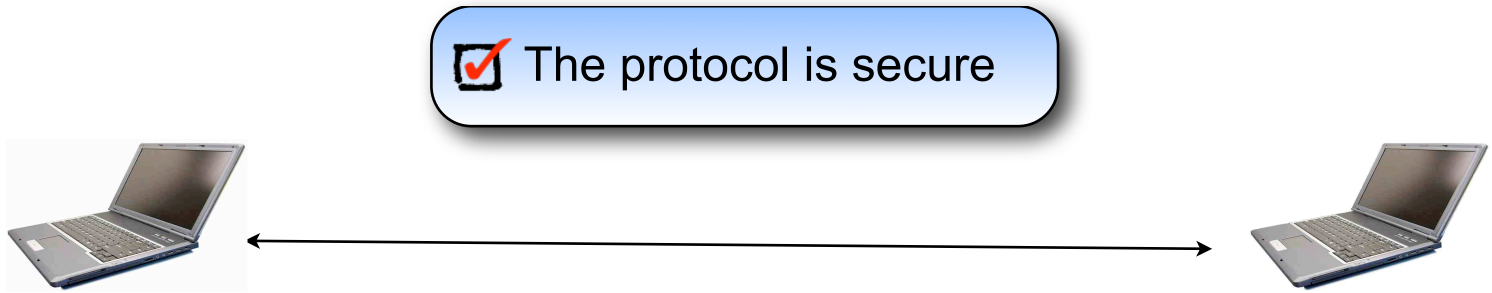
Type System

- Type-checking fully automated and efficient
- Compositional therefore scalable



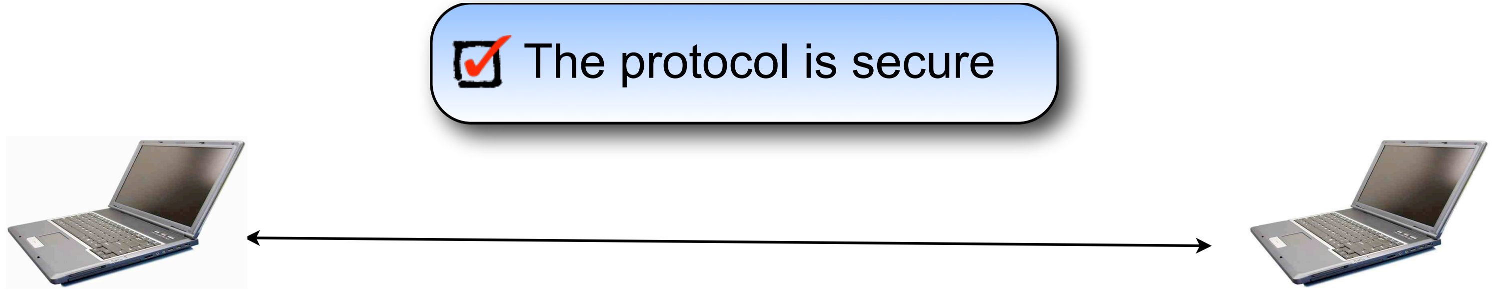
Type System

- Type-checking fully automated and efficient
- Compositional therefore scalable



Type System

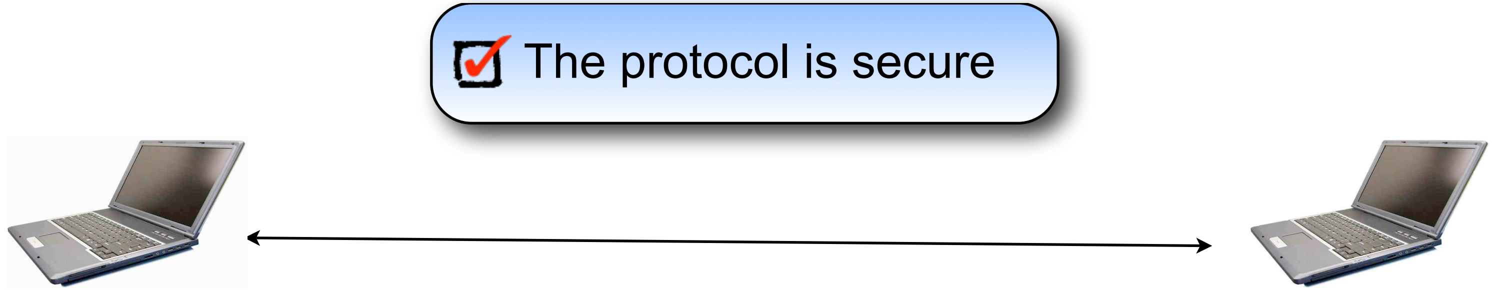
- Type-checking fully automated and efficient
- Compositional therefore scalable



- Predictable termination behavior

Type System

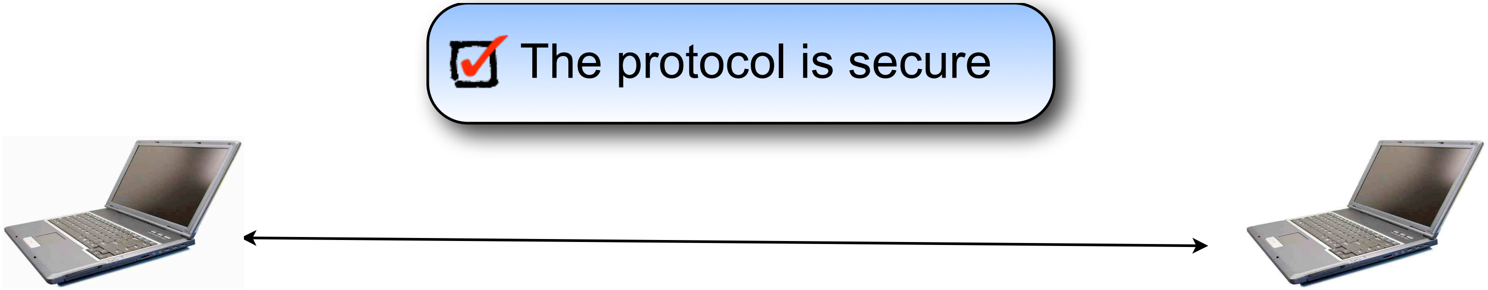
- Type-checking fully automated and efficient
- Compositional therefore scalable



- Predictable termination behavior
- User needs to provide annotations (no free lunch)

Type System

- Type-checking fully automated and efficient
- Compositional therefore scalable

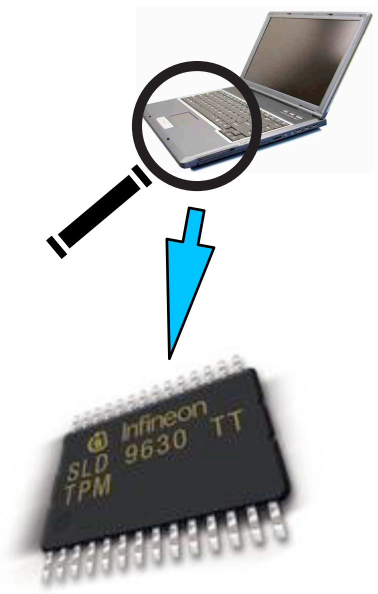


- Predictable termination behavior
- User needs to provide annotations (no free lunch)
  - in certain cases these can be automatically inferred

Type-checking DAA

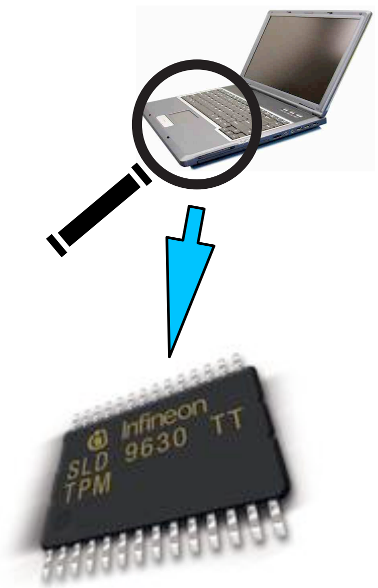
(Direct Anonymous Attestation)

DAA (Direct Anonymous Attestation)



[Brickell, Camenisch & Chen, CCS 2004]

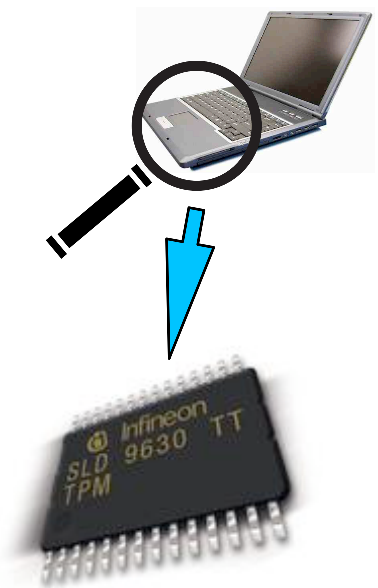
DAA (Direct Anonymous Attestation)



The user wants to authenticate a message *m* by proving that her platform has a valid TPM inside (*attestation*) ...

[Brickell, Camenisch & Chen, CCS 2004]

DAA (Direct Anonymous Attestation)

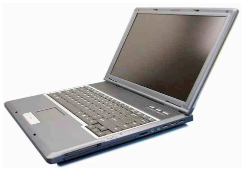


The user wants to authenticate a message *m* by proving that her platform has a valid TPM inside (*attestation*) ...

... but no other party should learn *which* TPM is used to authenticate *m* (*anonymity*)

[Brickell, Camenisch & Chen, CCS 2004]

Direct Anonymous Attestation (DAA)



f_{tpm}
(secret TPM identifier)

Issuer

Direct Anonymous Attestation (DAA)

Issuer

Blind signature on f_{tpm}

Joining Protocol

The user receives a certificate of f_{tpm} from the issuer

Direct Anonymous Attestation (DAA)

Issuer

$\text{sign}(f_{\text{tpm}}, k_I)$

Joining Protocol

The user receives a certificate of f_{tpm} from the issuer

Direct Anonymous Attestation (DAA)

Issuer

$\mathsf{sign}(f_{\mathsf{tpm}}, k_I)$

Zero-knowledge proof

Verifier

Signing Protocol

The user proves the knowledge of a certificate for his secret TPM identifier f_{tpm} ... without revealing it!

Direct Anonymous Attestation (DAA)

Issuer

$$\mathsf{sign}(f_{\mathsf{tpm}}, k_I)$$

Verifier

Zero-knowledge proof

"the user knows a secret identifier and a certificate, and the certificate is a valid signature made by the issuer on the identifier"

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathsf{tpm}}, k_I)$

Zero-knowledge proof

Verifier

"the user knows a secret identifier and a certificate, and the certificate is a valid signature made by the issuer on the identifier"

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

sign(f_{tpm}, k_I)

Zero-knowledge proof

non-interactive

Verifier

"the user knows a secret identifier and a certificate, and the certificate is a valid signature made by the issuer on the identifier"

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\text{sign}(f_{\text{tpm}}, k_I)$

Zero-knowledge proof

Verifier

$$\text{zk}_{\text{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\text{tpm}}, \text{sign}(f_{\text{tpm}}, k_I);\ \text{vk}(k_I), m)$$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\text{sign}(f_{\text{tpm}}, k_I)$

Zero-knowledge proof

Verifier

secret witnesses

$$\text{zk}_{\text{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\text{tpm}}, \text{sign}(f_{\text{tpm}}, k_I);\ \text{vk}(k_I), m)$$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\text{tpm}}, k_I)$

Zero-knowledge proof

Verifier

public messages

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathrm{tpm}}, k_I)$

Zero-knowledge proof

Verifier

statement

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

$$\mathsf{chk}(\quad \alpha_2 \quad , \quad \beta_1 \quad) = \alpha_1$$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathrm{tpm}}, k_I)$

Zero-knowledge proof

Verifier

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

Verification: $\mathsf{chk}(\quad \alpha_2 \quad , \quad \beta_1 \quad) = \alpha_1$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathrm{tpm}}, k_I)$

Zero-knowledge proof

Verifier

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

Verification: $\mathsf{chk}(\mathsf{sign}(f_{\mathrm{tpm}}, k_I),\quad \beta_1\quad) = \alpha_1$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathrm{tpm}}, k_I)$

Zero-knowledge proof

Verifier

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

Verification: $\mathsf{chk}(\mathsf{sign}(f_{\mathrm{tpm}}, k_I), \mathsf{vk}(k_I)) = \alpha_1$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

sign(f_{tpm}, k_I)

Zero-knowledge proof

Verifier

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

Verification: $\mathsf{chk}(\mathsf{sign}(f_{\mathrm{tpm}}, k_I), \mathsf{vk}(k_I)) = f_{\mathrm{tpm}}$

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

Idealization of Zero-knowledge

Prover

$\mathsf{sign}(f_{\mathrm{tpm}}, k_I)$

Zero-knowledge proof

Verifier

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\ \mathsf{vk}(k_I), m)$$

Verification: $\mathsf{chk}(\mathsf{sign}(f_{\mathrm{tpm}}, k_I), \mathsf{vk}(k_I)) = f_{\mathrm{tpm}}$ ✓

[Backes, Maffei & Unruh, S&P 2008] [Backes & Unruh, CSF 2008]

DAA Signing Protocol (simplified)

Prover

$$\xrightarrow{\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)}$$

Verifier

$$\begin{array}{l} \mathsf{new}\ k_I \\ \mathsf{new}\ f_{\mathrm{tpm}} \\ \mathrm{Prover}\ \mid\ \mathrm{Verifier}\ \mid\ \mathrm{Issuer} \end{array}$$

$$\begin{array}{lcl} \mathrm{Prover} & = & \mathsf{new}\ m \\ & & \mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)) \end{array}$$

$$\begin{array}{lcl} \mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \end{array}$$

DAA Signing Protocol (simplified)

Prover

$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

Verifier

$\mathsf{new}\ k_I$
$\mathsf{new}\ f_{\mathrm{tpm}}$
$\mathrm{Prover}\ \mid\ \mathrm{Verifier}\ \mid\ \mathrm{Issuer}$

$$\mathrm{Prover} \;=\; \mathsf{new}\ m$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

$$\mathrm{Verifier} \;=\; \mathsf{in}(c, x).$$

$$\mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then}$$

matching will help typing

Security Annotations

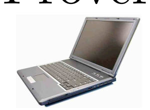
Prover

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$$

Verifier

$\mathsf{new}\ k_I$
$\mathsf{new}\ f_{\mathrm{tpm}}$
$\mathrm{Prover} \ \mid\ \mathrm{Verifier} \ \mid\ \mathrm{Issuer}$

$\mathrm{Prover} \ = \ \mathsf{new}\ m$



$\mathsf{assume}\ \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid$

$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$

$\mathrm{Verifier} \ = \ \mathsf{in}(c, x).$

$\mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then}$

Security Annotations

Prover

$\xrightarrow{\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},\, \mathsf{sign}(f_{\mathrm{tpm}}, k_I);\, \mathsf{vk}(k_I), m)}$

Verifier

$\mathsf{assume}\ \forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m) \quad |$

$\mathsf{new}\ k_I$

$\mathsf{new}\ f_{\mathrm{tpm}}$

$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$

authorization policy
(in some logic)

$$
\begin{aligned}
\mathrm{Prover} \quad = \quad & \mathsf{new}\ m \\
& \mathsf{assume}\ \mathrm{Send}(f_{\mathrm{tpm}}, m) \ | \\
& \mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Verifier} \quad = \quad & \mathsf{in}(c, x). \\
& \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\
& \mathsf{assert}\ \mathrm{Authenticate}(y_m)
\end{aligned}
$$

Security Annotations

Prover

$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

Verifier

$\mathsf{assume}\ \forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m) \quad |$

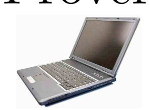
$\mathsf{new}\ k_I$

$\mathsf{new}\ f_{\mathrm{tpm}}$

$\mathrm{Prover} \ | \ \mathrm{Verifier} \ | \ \mathrm{Issuer}$

authorization policy (in some logic)

$\mathrm{Prover} \ = \ \mathsf{new}\ m$



$\mathsf{assume}\ \mathrm{Send}(f_{\mathrm{tpm}}, m) \ |$

$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$

$\mathrm{Verifier} \ = \ \mathsf{in}(c, x).$

$\mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))$

$\mathsf{assert}\ \mathrm{Authenticate}(y_m)$

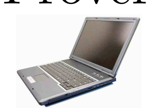
Safety

Asserts are entailed by the current assumes

Basic Types

assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)$ $|$
new k_I
new f_{tpm}
$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$

$\mathrm{Prover} \;=\;$ new m
assume $\mathrm{Send}(f_{\mathrm{tpm}}, m)$ $|$
$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$



$\mathrm{Verifier} \;=\;$ $\mathsf{in}(c, x)$.
let $y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))$ then
assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$
new k_I
new f_{tpm}
$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$

$\mathrm{Prover} \quad =$ new m: Un

assume Send(
out$(c,$ zk$_{\mathsf{chk}(}$ ... $_I);$ vk$(k_I), m))$

Type of messages known to the attacker

$\mathrm{Verifier} \quad =$ in(c, x).

let $y_m =$ ver$_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}$ ... $(k_I))$ then
assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$

new k_I

new f_{tpm}: Private

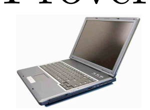
$\mathrm{Prover} \quad | \quad \mathrm{Verifier}$

Type of messages unknown to the attacker

$\mathrm{Prover} \quad =$ new m

assume S

out$(c,$ zk$_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},$ sign$(f_{\mathrm{tpm}}, k_I);$ vk$(k_I), m))$



$\mathrm{Verifier} \quad =$ in(c, x).

let $y_m =$ ver$_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x;$ vk$(k_I))$ then

assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$
new k_I: SigKey($\langle x_f$: Private$\rangle\{\mathrm{OkTPM}(x_f)\}$)
new f_{tpm}: Private
$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issue}$

Type of keys used
to sign Private messages for which
OkTPM holds

$\mathrm{Prover} \quad =$ new m: Un

assume $\mathrm{Send}(f_{\mathrm{tpm}}, m) \mid$
out$(c,$ zk$_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},$ sign(f_{tpm}, k_I); vk$(k_I), m))$

$\mathrm{Verifier} \quad =$ in(c, x).

let $y_m =$ ver$_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x;$ vk$(k_I))$ then
assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

The type of the key allows us to “transfer” predicates from the prover to the verifier!

assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$
new k_I: $\mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\})$
new f_{tpm}: $\mathsf{Private}$
$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$

$\mathrm{Prover} \quad = \quad$ new m: Un
assume $\mathrm{Send}(f_{\mathrm{tpm}}, m) \ |$
$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2, \beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$

$\mathrm{Verifier} \quad = \quad \mathsf{in}(c, x).$
let $y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2, \beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))$ then
assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

But, the verifier can't use the key to check a certificate he never receives.

$$\text{assume } \forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \land \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$$
$$\text{new } k_I \colon \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\})$$
$$\text{new } f_{\mathrm{tpm}} \colon \mathsf{Private}$$
$$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$$

$$\mathrm{Prover} \quad = \quad \text{new } m \colon \mathsf{Un}$$
$$\text{assume } \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid$$
$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

$$\mathrm{Verifier} \quad = \quad \mathsf{in}(c, x).$$
$$\text{let } y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I)) \text{ then}$$
$$\text{assert } \mathrm{Authenticate}(y_m)$$

[Fournet, Gordon & Maffeis, CSF 2007]

Basic Types

But, the verifier can't use the key to check a certificate he never receives.

Worse, ZK don't necessarily rely on keys!

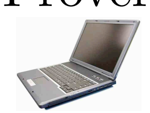
assume $\forall m.\ (\exists x_f.\ \mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m) \quad |$

new k_I: SigKey($\langle x_f$: Private$\rangle\{\mathrm{OkTPM}(x_f)\}$)

new f_{tpm}: Private

$\mathrm{Prover} \quad | \quad \mathrm{Verifier} \quad | \quad \mathrm{Issuer}$

$\mathrm{Prover} \quad =$ new m: Un



assume $\mathrm{Send}(f_{\mathrm{tpm}}, m)$ |

out$(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$

$\mathrm{Verifier} \quad =$ in(c, x).

let $y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))$ then

assert $\mathrm{Authenticate}(y_m)$

[Fournet, Gordon & Maffeis, CSF 2007]

Typing Zero-knowledge Proofs

Our solution:

User gives a type to each statement proved by ZK

Typing Zero-knowledge Proofs

Prover

$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\rm tpm}, \mathsf{sign}(f_{\rm tpm}, k_I); \mathsf{vk}(k_I), m)$

Verifier

$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(\langle y_k : \mathsf{VerKey}(\ldots), y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\})$

Typing Zero-knowledge Proofs

Prover

$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

Verifier

Type of
public messages

$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(\langle y_k : \mathsf{VerKey}(\ldots), y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\})$

Typing Zero-knowledge Proofs

Prover

$$\mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$$

Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(\langle y_k : \mathsf{VerKey}(\ldots), y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\})$$

Logical formula where the secret witnesses are *existentially quantified*

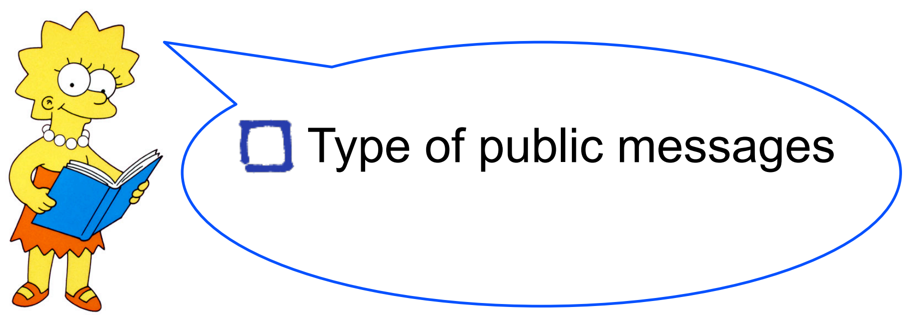
Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{lll}\mathrm{Prover} & = & \ldots \\ & & \mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))\end{array}$$

Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$



$$\mathrm{Prover} \quad = \quad \ldots$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{aligned}\Gamma = \;& \ldots \\ & k_I \colon \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & m : \mathsf{Un}, \\ & \ldots \\ & \mathrm{OkTPM}(f_{\mathrm{tpm}}), \\ & \mathrm{Send}(f_{\mathrm{tpm}}, m)\end{aligned}$$

☐ Type of public messages

$$\mathrm{Prover} \;=\; \ldots$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rl}\Gamma = & \ldots \\ & k_I \colon \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & m : \mathsf{Un}, \\ & \ldots \\ & \mathrm{OkTPM}(f_{\mathrm{tpm}}), \\ & \mathrm{Send}(f_{\mathrm{tpm}}, m)\end{array}$$

☑ Type of public messages
☐ Logical formula entailed

$$\mathrm{Prover} \;=\; \ldots$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{ll}\Gamma = & \ldots \\ & k_I\text{: } \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & m : \mathsf{Un}, \\ & \ldots \\ & \mathrm{OkTPM}(f_{\mathrm{tpm}}), \\ & \mathrm{Send}(f_{\mathrm{tpm}}, m)\end{array}$$

☑ Type of public messages
☑ Logical formula entailed

$$\mathrm{Prover} \;=\; \ldots$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

Type-checking the Prover

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{aligned}\Gamma = \;& \dots \\ & k_I\!: \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & m : \mathsf{Un}, \\ & \dots \\ & \mathrm{OkTPM}(f_{\mathrm{tpm}}), \\ & \mathrm{Send}(f_{\mathrm{tpm}}, m)\end{aligned}$$

$$\mathrm{Prover} \;=\; \dots$$

$$\mathsf{out}(c, \mathsf{zk}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)),\end{array}$$

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)),\end{array}$$

If verification succeeds, can we assume the formula in ZK type?

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{aligned}\Gamma \;=\;\; & \ldots \\ & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & \ldots \\ & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)),\end{aligned}$$

If verification succeeds, can we assume the formula in ZK type?

In general not, the proof can come from untyped adversary!

$$\begin{aligned}\mathrm{Verifier} \;=\;\; & \mathsf{in}(c, x). \\ & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{aligned}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Conceptual issue

How can we know whether the zero-knowledge proof comes from an honest participant or from the adversary?!

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists \boldsymbol{x_f}, \boldsymbol{x_s}.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)),\end{array}$$

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists \boldsymbol{x_f}, \boldsymbol{x_s}.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$\Gamma \;=\; \ldots$

$k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTP}$

$\ldots$

$\forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTP}$

$\mathsf{chk}(\alpha_2, \quad \beta_1 \quad) = \quad \alpha_1$

$\mathrm{Verifier} \;=\; \mathsf{in}(c, x).$

$\mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then}$

$\mathsf{assert}\ \mathrm{Authenticate}(y_m)$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists \boldsymbol{x_f}, \boldsymbol{x_s}.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{lll} \Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM} \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTP} \\ & & \mathsf{chk}(\alpha_2, \quad \beta_1 \quad) = \alpha_1 \end{array}$$

The statement
instantiated with the matched
messages is valid
(by the semantics)

$$\begin{array}{lll} \mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m) \end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM} \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPI} \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f\end{array}$$

The statement
instantiated with the matched
messages is valid
(by the semantics)

$$\begin{array}{rcl}\text{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m)), \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & & x_f : \mathsf{Private}\end{array}$$

The type of vk(k_I) gives us the type of x_f (existentially quantified)

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}) \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & & x_f : \mathsf{Private}\end{array}$$

The prover is honest, since he knows a message of type Private!

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rl}\Gamma \; = & \ldots \\ & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}) \\ & \ldots \\ & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x \\ & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & x_f : \mathsf{Private}\end{array}$$

We can now exploit the formula in the ZK type

$$\begin{array}{rl}\mathrm{Verifier} \; = & \mathsf{in}(c, x). \\ & \mathsf{let}\; y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\; \mathsf{then} \\ & \mathsf{assert}\; \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl} \Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)), \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & & x_f : \mathsf{Private} \\ & & \exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f) \end{array}$$

$$\begin{array}{rcl} \mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m) \end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{lll}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)), \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & & x_f : \mathsf{Private} \\ & & \exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\end{array}$$

$$\begin{array}{lll}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots\\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}),\\ & & \ldots\\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)),\\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f\\ & & x_f : \mathsf{Private}\\ & & \exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\end{array}$$

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x).\\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then}\\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the Verifier

$$\mathsf{ZK}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$$\begin{array}{rcl}\Gamma & = & \ldots \\ & & k_I : \mathsf{SigKey}(\langle x_f : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x_f)\}), \\ & & \ldots \\ & & \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m)), \\ & & \mathsf{chk}(x_s, \mathsf{vk}(k_I)) = x_f \\ & & x_f : \mathsf{Private} \\ & & \exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\end{array}$$

$$\begin{array}{rcl}\mathrm{Verifier} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ y_m = \mathsf{ver}_{\mathsf{chk}(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Take Home

- ZK proofs are given *dependent types* where the witnesses are *existentially quantified*

Take Home

- ZK proofs are given *dependent types* where the witnesses are *existentially quantified*
- *The prover* can only prove statements for which the formula in the ZK type holds

Take Home

- ZK proofs are given *dependent types* where the witnesses are *existentially quantified*
- *The prover* can only prove statements for which the formula in the ZK type holds
- *The verifier* can assume the formula in the ZK type

Take Home

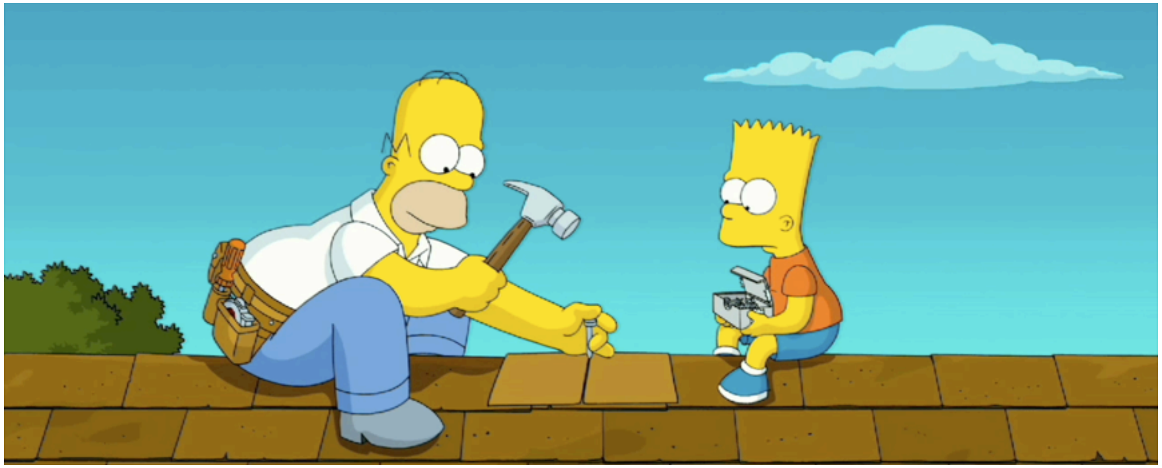
- ZK proofs are given *dependent types* where the witnesses are *existentially quantified*
- *The prover* can only prove statements for which the formula in the ZK type holds
- *The verifier* can assume the formula in the ZK type
  - if the formula is entirely derived from the proved statement (most often much too weak)

Take Home

- ZK proofs are given *dependent types* where the witnesses are *existentially quantified*
- *The prover* can only prove statements for which the formula in the ZK type holds
- *The verifier* can assume the formula in the ZK type
  - if the formula is entirely derived from the proved statement (most often much too weak)
  - if he can somehow infer that the proof was constructed by an *honest prover (type-checked)*

Implementation

- Type-checker written in O'Caml (~5000 LOC)
- Uses automatic prover for discharging FOL formulas
- Extensible - very easy to add arbitrary primitives + types
- Efficient - the complete analysis of DAA takes 0.7s
- Available under the Apache License: http://www.infsec.cs.uni-sb.de/projects/zk-typechecker/
- Kudos to Stefan Lorenz, Kim Pecina and Thorsten Tarrach



Ongoing Work

- *Type-checking a model of Civitas*
  - Remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]

Ongoing Work

- *Type-checking a model of Civitas*
  - Remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]
- *Type-checking implementations of protocols that employ zero-knowledge*
  - Trying to extend [Bengtson et al., CSF 2008]

Ongoing Work

- *Type-checking a model of Civitas*
  - Remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]
- *Type-checking implementations of protocols that employ zero-knowledge*
  - Trying to extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise*
  - Execute original protocol, but add zero-knowledge to prove correct behavior to remote parties

Ongoing Work

- *Type-checking a model of Civitas*
  - Remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]
- *Type-checking implementations of protocols that employ zero-knowledge*
  - Trying to extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise*
  - Execute original protocol, but add zero-knowledge to prove correct behavior to remote parties
- *Idealizing interactive zero-knowledge proofs and analyzing the protocols that use them*

Ongoing Work

- *Type-checking a model of Civitas*
  - Remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]
- *Type-checking implementations of protocols that employ zero-knowledge*
  - Trying to extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise*
  - Execute original protocol, but add zero-knowledge to prove correct behavior to remote parties
- *Idealizing interactive zero-knowledge proofs and analyzing the protocols that use them*

THANK YOU!