IS&C

UNIVERSITÄT DES SAARLANDES

Type-checking Zero-knowledge

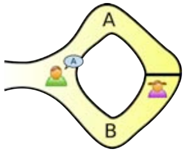
Cătălin Hrițcu

Saarland University, Saarbrücken, Germany

Joint work with: Michael Backes and Matteo Maffei

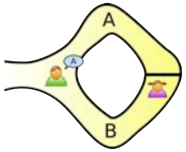
Microsoft Research Cambridge, July 2008

Zero-knowledge proofs



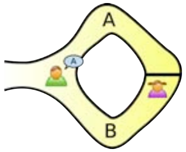
- Powerful cryptographic primitives
  - Prove the existence of an object with certain properties without revealing this object to anyone

Zero-knowledge proofs



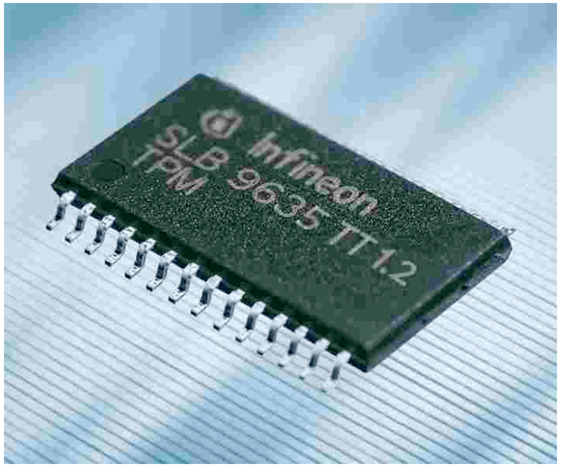
- Powerful cryptographic primitives
  - Prove the existence of an object with certain properties without revealing this object to anyone
- Early constructions very general
  - But terribly inefficient
  - Very limited practical impact

Zero-knowledge proofs



- Powerful cryptographic primitives
  - Prove the existence of an object with certain properties without revealing this object to anyone
- Early constructions very 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

Many emerging applications use ZK



Lack of verification tools for ZK

- When we started this, there were no automated verification tools for protocols using zero-knowledge proofs as a primitive

Lack of verification tools for ZK

- When we started this, there were no automated verification tools for protocols using zero-knowledge proofs as a primitive
- Security protocols are hard to get right

Lack of verification tools for ZK

- When we started this, there were no automated verification tools for protocols using zero-knowledge proofs as a primitive
- Security protocols are hard to get right
- Automated verification can really help protocol designers prevent high-level errors

Lack of verification tools for ZK

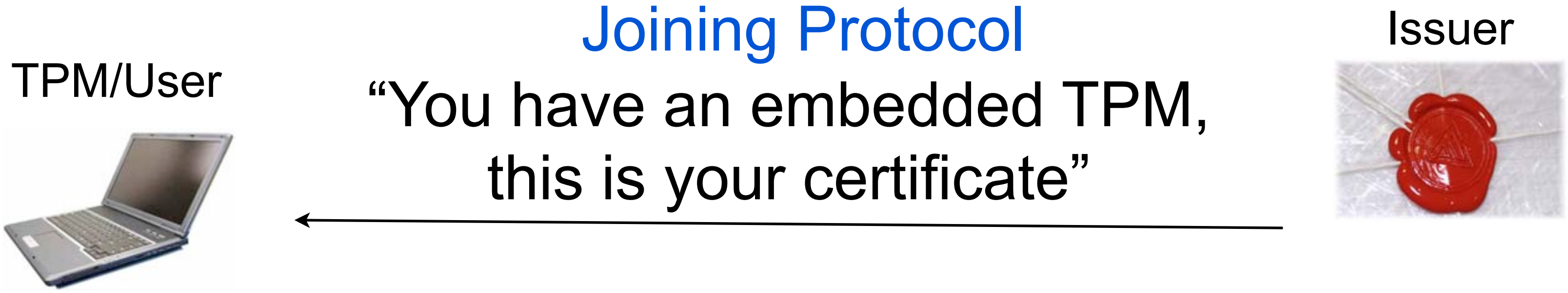
- When we started this, there were no automated verification tools for protocols using zero-knowledge proofs as a primitive
- Security protocols are hard to get right
- Automated verification can really help protocol designers prevent high-level errors
- We provided two ways to automatically analyze protocols using zero-knowledge
  - Using ProVerif [Backes, Maffei & Unruh, S&P 2008]
  - Using a type system [Backes, Hrițcu & Maffei, CCS 2008]

Outline

- Zero-knowledge proofs at work
  - *Direct Anonymous Attestation (DAA)* protocol (extremely simplified in my example)
- *Modeling zero-knowledge proofs symbolically*
- *Type system* to statically enforce authorization policies for protocols using zero-knowledge proofs
  - Extension of [Fournet, Gordon & Maffeis, CSF 2007]

Direct Anonymous Attestation (DAA)

Joining Protocol

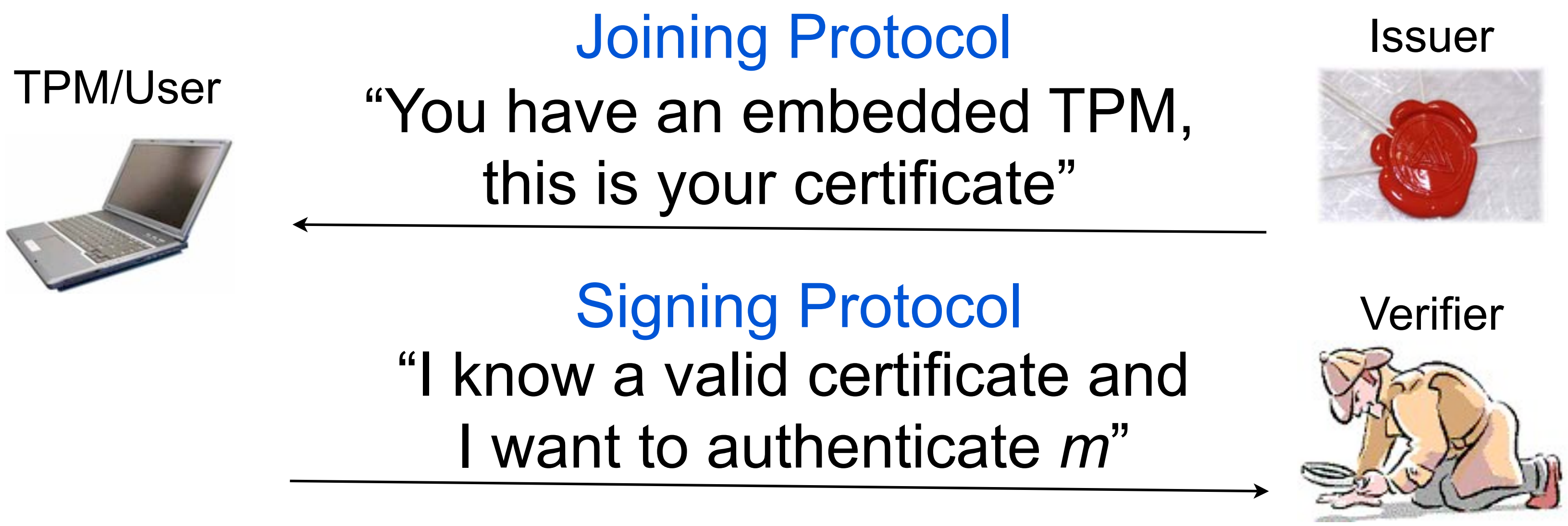


TPM/User

Issuer

“You have an embedded TPM, this is your certificate”

Direct Anonymous Attestation (DAA)



TPM/User

Issuer

Joining Protocol

“You have an embedded TPM, this is your certificate”

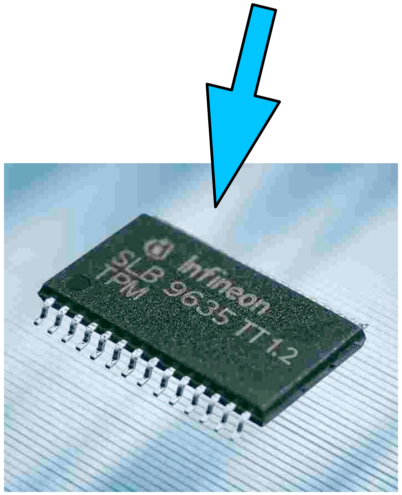
Verifier

Signing Protocol

“I know a valid certificate and I want to authenticate *m*”

Direct Anonymous Attestation (DAA)

TPM/User



Joining Protocol

“You have an embedded TPM, this is your certificate”

Issuer

Signing Protocol

“I know a valid certificate and I want to authenticate *m*”

Verifier

The user proves that her platform has a valid TPM inside (*attestation*)...

Direct Anonymous Attestation (DAA)

TPM/User

Joining Protocol

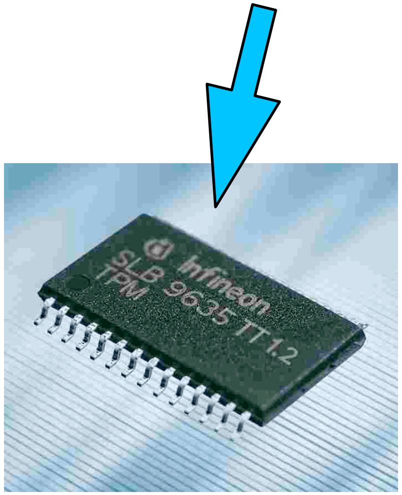
"You have an embedded TPM, this is your certificate" ←

Issuer

Signing Protocol

"I know a valid certificate and I want to authenticate *m*" →

Verifier



The user proves that her platform has a valid TPM inside (*attestation*)...

... but the other parties do not learn *which* TPM is used to authenticate m (*anonymity*)

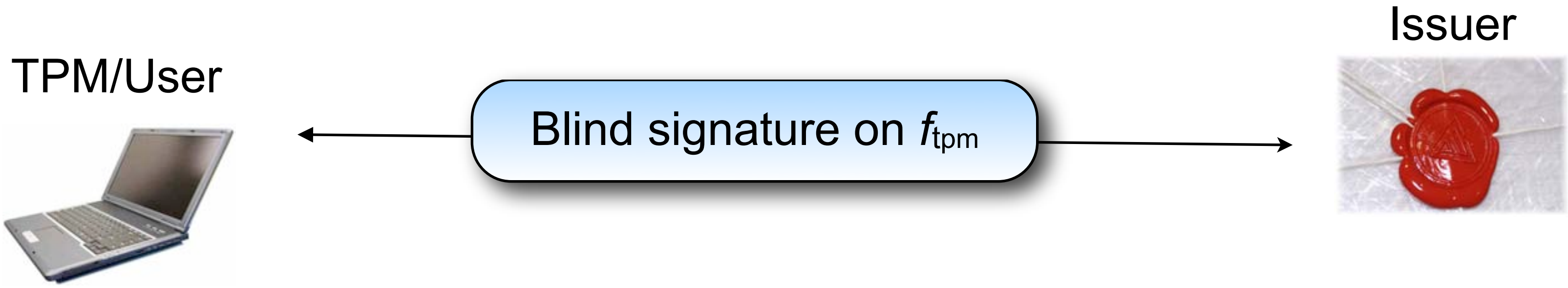
Direct Anonymous Attestation (DAA)

TPM/User

f_{tpm}
(secret TPM identifier)

Issuer

Direct Anonymous Attestation (DAA)



TPM/User

Blind signature on f_{tpm}

Issuer

Joining Protocol

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

Direct Anonymous Attestation (DAA)

TPM/User

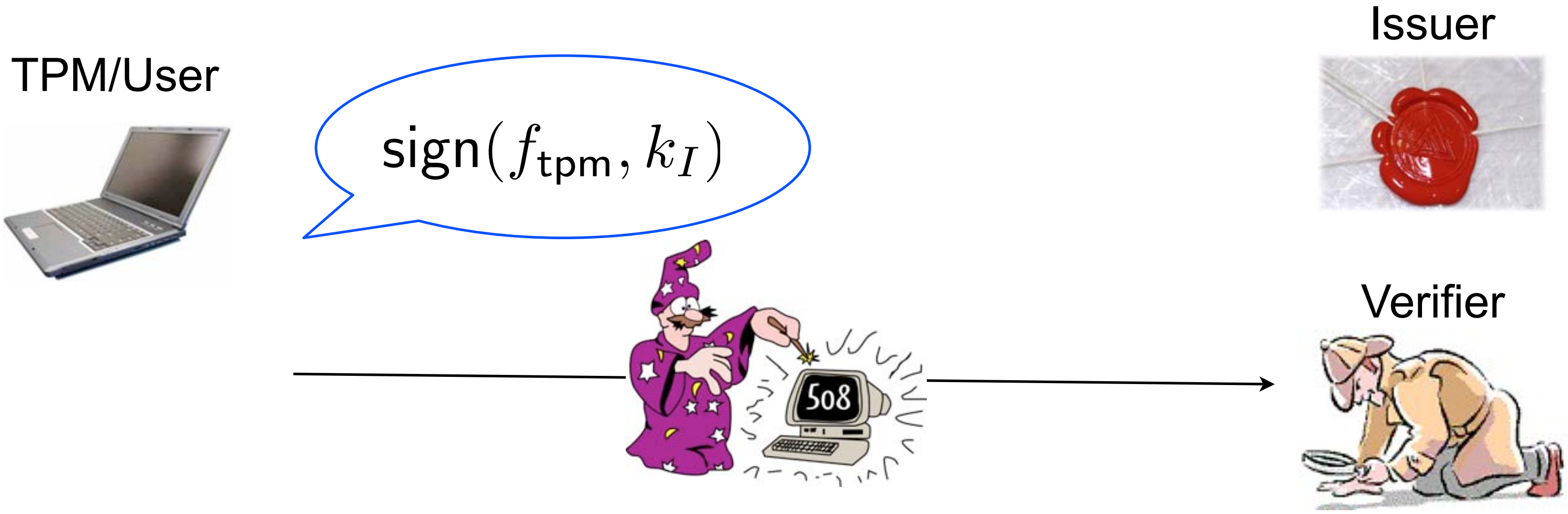
$\mathsf{sign}(f_{\mathsf{tpm}}, k_I)$

Issuer

Joining Protocol

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

Direct Anonymous Attestation (DAA)



TPM/User

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

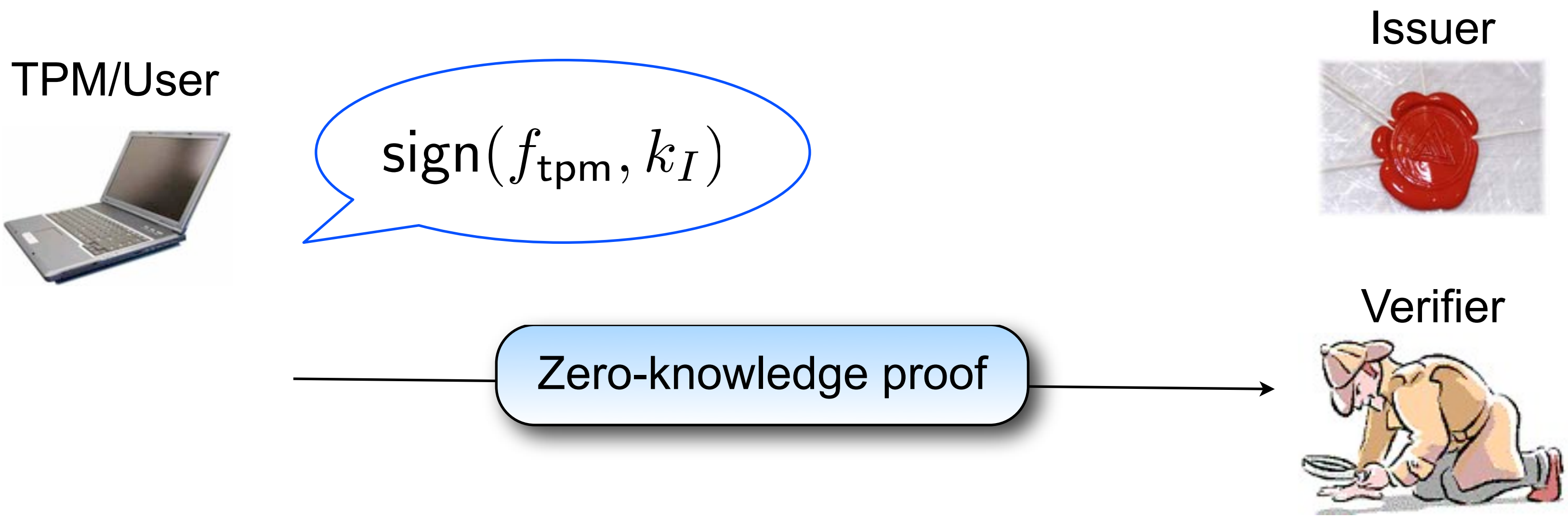
Issuer

Verifier

Signing Protocol

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

Direct Anonymous Attestation (DAA)

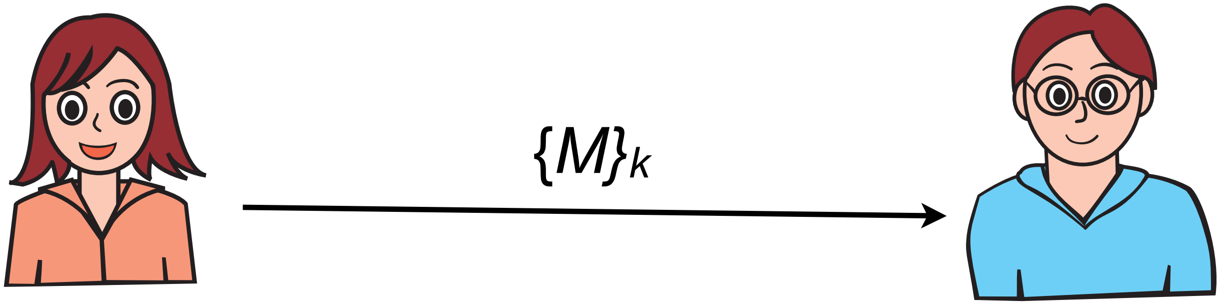


“there exists a secret α_f and a certificate α_{sign} such that the verification of α_{sign} with vk(k_I) succeeds and the content of α_{sign} is α_f

Modeling zero-knowledge proofs symbolically

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]



out(*ch*, enc(*M* , *k*)).

...

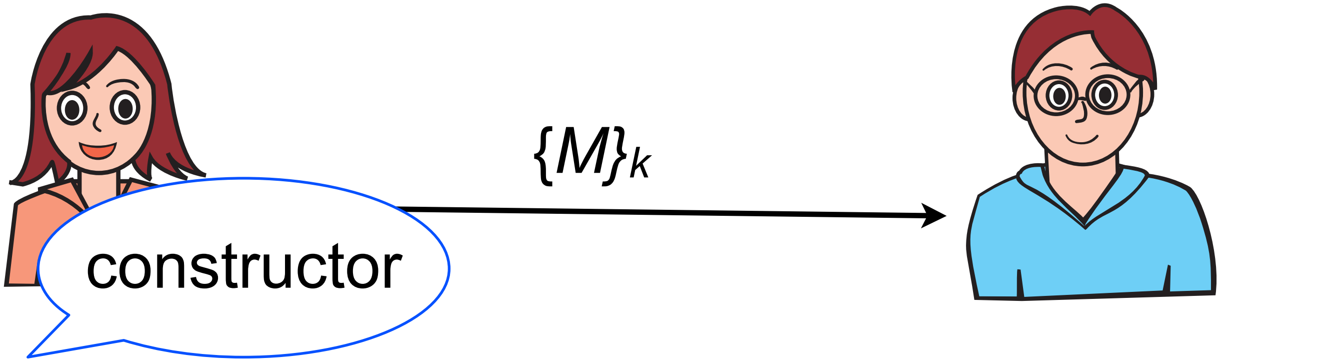
in(*ch*, *x*).

let y = dec (*x* , *k*)

...

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]



out(*ch*, enc(*M* , *k*)).

...

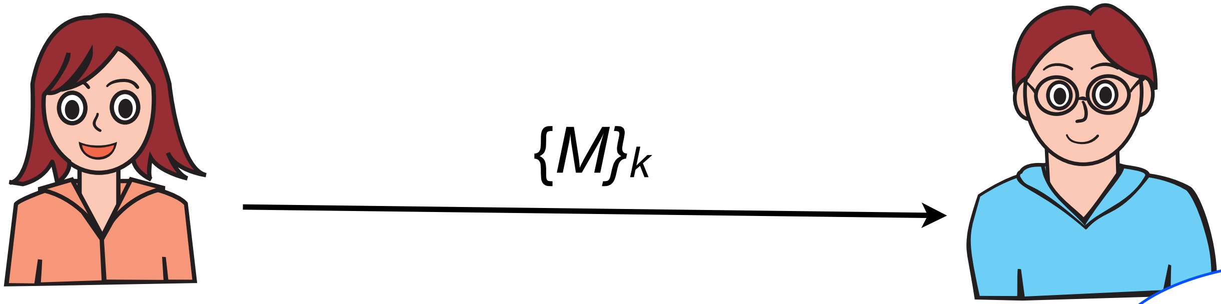
in(*ch*, *x*).

let y = dec (*x* , *k*)

...

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]



out(*ch*, enc(*M* , *k*)).

...

in(*ch*, *x*

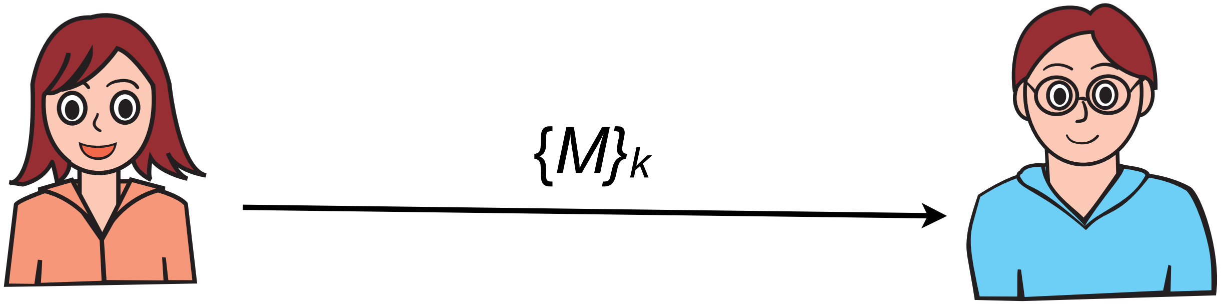
destructor

let y = dec (*x* , *k*)

...

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]



out(*ch*, enc(*M* , *k*)).

...

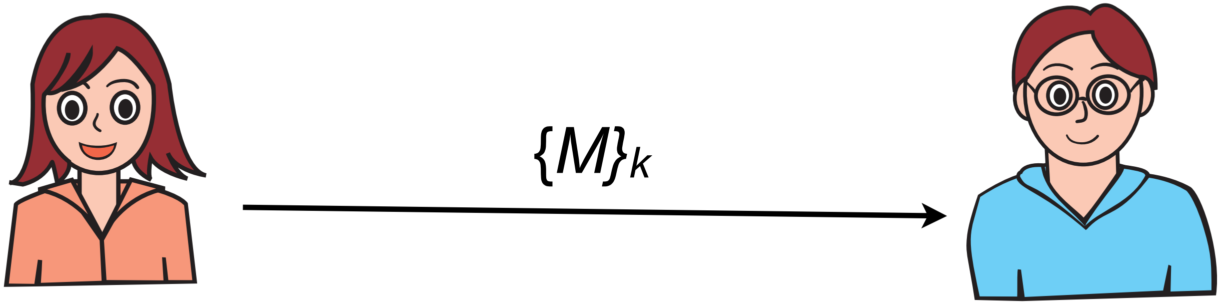
in(*ch*, *x*).

let y = dec (*x* , *k*)

...

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]

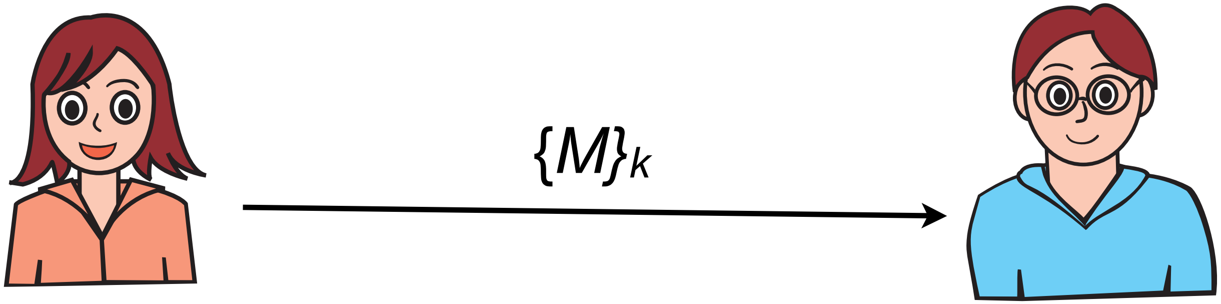


...

in(ch, enc(M , k)).
let y = dec (x , k)
...

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]

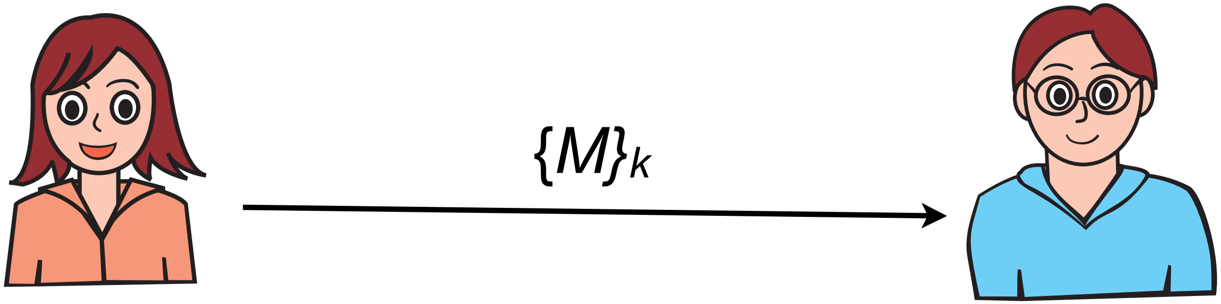


...

let y = dec (enc(M , k) , k) then

An extensible spi-calculus

Cryptographic primitives modeled as
user-defined constructors and *destructors*
[Abadi & Blanchet 2002]



... let y = dec (enc(M , k) , k) then

Reduction relation ⇓

Reduction relation for destructors

The semantics of the calculus is parameterized by a user-defined reduction relation for destructors:

Crypto

$\mathrm{dec}(\ \mathrm{enc}(x,y)\ ,\ y\ ) \Downarrow\ x$

$\mathrm{chk}(\mathrm{sign}(x,y)\ ,\ \mathrm{vk}(y)) \Downarrow x$

Data

$\mathrm{first}(\mathrm{pair}(x,y)) \Downarrow \mathrm{x}$

$\mathrm{snd}(\mathrm{pair}(x,y))\ \Downarrow \mathrm{y}$

$\mathrm{eq}(x,x) \quad \Downarrow \mathrm{true}$

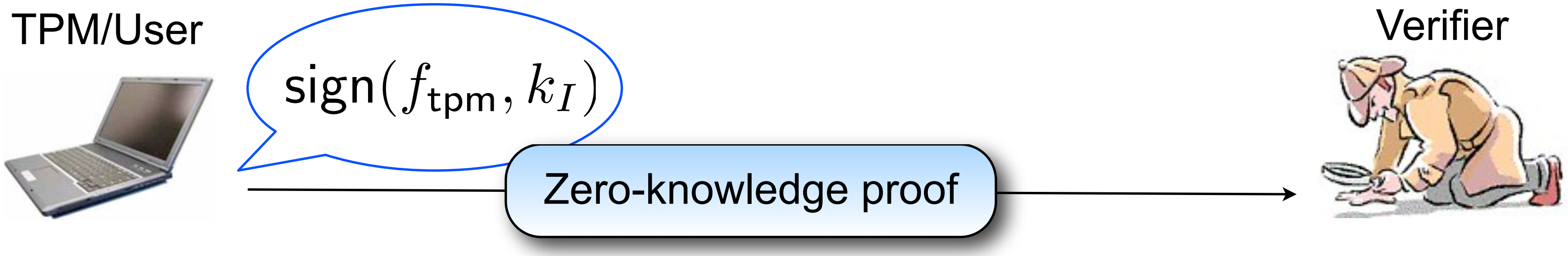
Logic

$\mathrm{and}(\mathrm{true},\ \mathrm{true}) \Downarrow \mathrm{true}$

$\mathrm{or}(x,\mathrm{true}) \quad \Downarrow \mathrm{true}$

$\mathrm{or}(\mathrm{true},x) \quad \Downarrow \mathrm{true}$

Abstraction of zero-knowledge



"there exists a secret α_f and a certificate α_{sign} such that the verification of α_{sign} with vk(k_I) succeeds and the content α_{sign} of is α_f

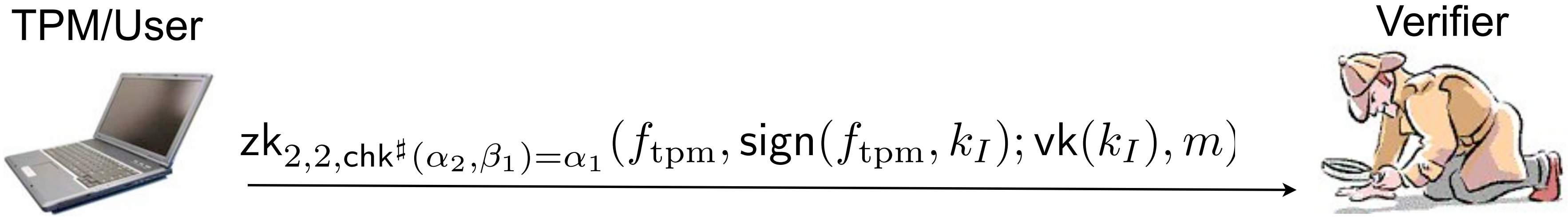
Abstraction of zero-knowledge

TPM/User

$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^{\sharp}(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)}$$

Verifier

Abstraction of zero-knowledge



$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}\Big(f_{\mathrm{tpm}} \;,\; \mathsf{sign}(f_{\mathsf{tpm}}, k_I) \;;\; \mathsf{vk}(k_I) \;,\; m\Big)$$

Abstraction of zero-knowledge

TPM/User

Verifier

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m) \longrightarrow$$

private messages

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}\Big(f_{\mathrm{tpm}} \;,\; \mathsf{sign}(f_{\mathsf{tpm}}, k_I) \;\;;\;\; \mathsf{vk}(k_I) \;,\; m\Big)$$

Abstraction of zero-knowledge

TPM/User

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m) \xrightarrow{\quad}$$

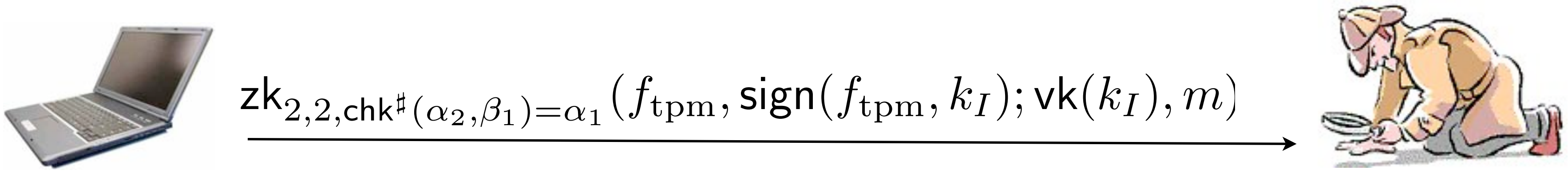
Verifier

private messages

public messages

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}\Big(\ f_{\mathrm{tpm}}\ ,\ \mathsf{sign}(f_{\mathsf{tpm}}, k_I)\ \ ;\ \ \mathsf{vk}(k_I)\ ,\ m\Big)$$

Abstraction of zero-knowledge



TPM/User

Verifier

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$$

statement

private messages

public messages

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}\Big(\; f_{\mathrm{tpm}} \;,\; \mathsf{sign}(f_{\mathrm{tpm}}, k_I) \;\;;\;\; \mathsf{vk}(k_I) \;,\; m \Big)$$

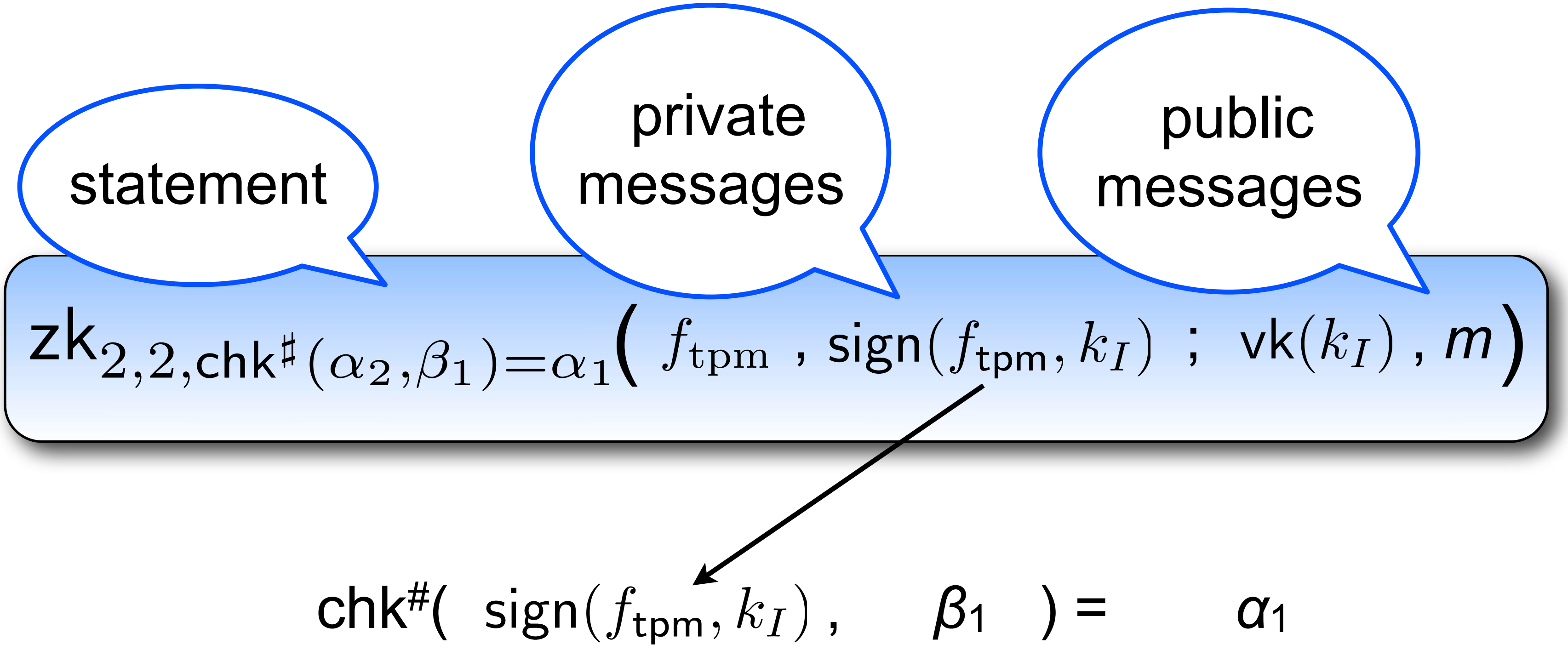
$$\mathsf{chk}^\#(\quad \alpha_2 \quad , \quad \beta_1 \quad) = \quad \alpha_1$$

Abstraction of zero-knowledge

TPM/User

$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

Verifier



statement

private messages

public messages

$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(\ f_{\mathrm{tpm}}\ ,\ \mathsf{sign}(f_{\mathrm{tpm}}, k_I)\ ;\ \mathsf{vk}(k_I)\ ,\ m)$

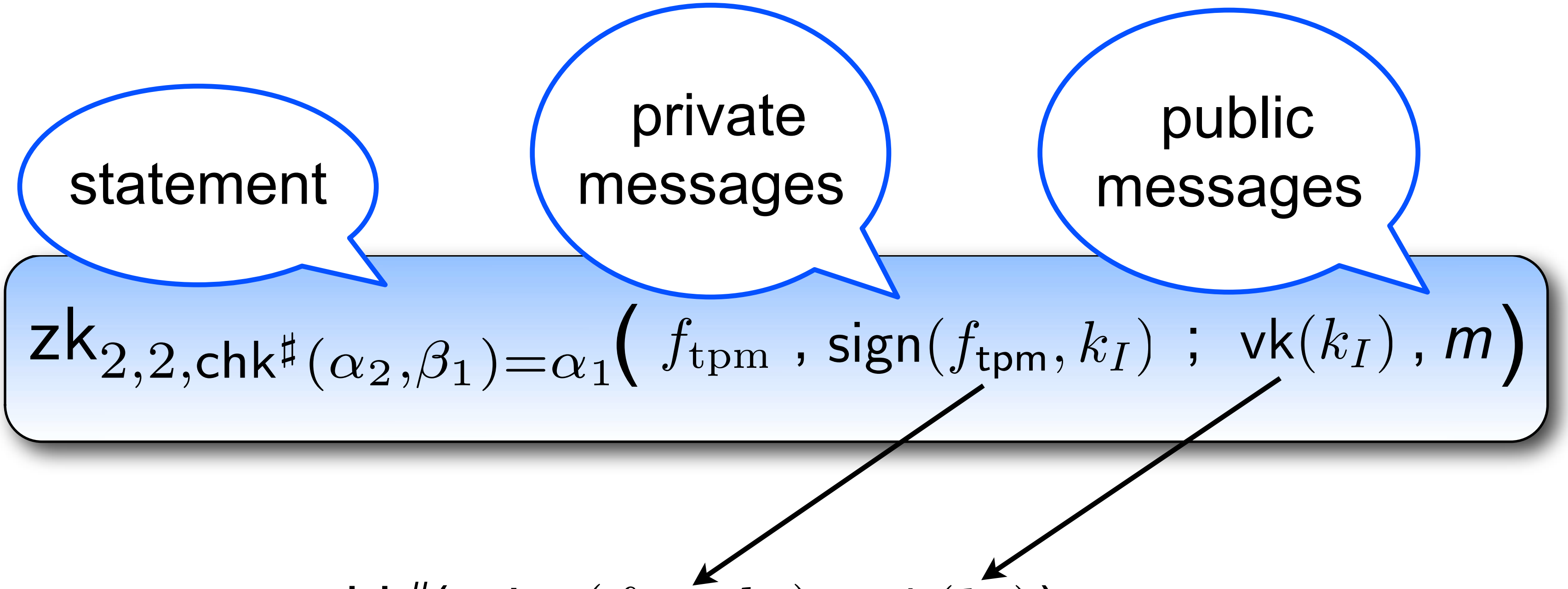
$\mathsf{chk}^{\#}(\ \mathsf{sign}(f_{\mathrm{tpm}}, k_I)\ ,\quad \beta_1\quad) = \quad \alpha_1$

Abstraction of zero-knowledge

TPM/User

Verifier

$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)}$$



$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}\big(f_{\mathrm{tpm}} , \mathsf{sign}(f_{\mathsf{tpm}}, k_I) ; \mathsf{vk}(k_I) , m \big)$$

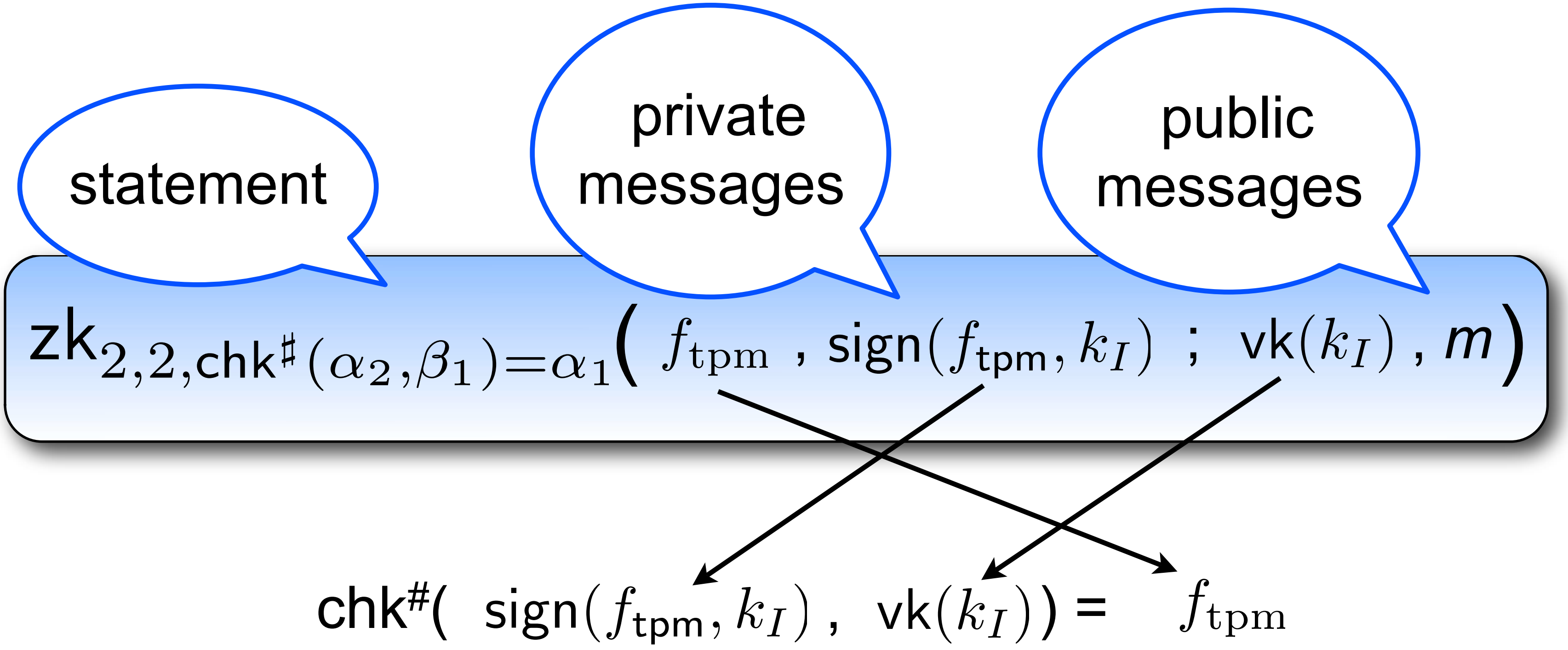
$$\mathsf{chk}^\#(\mathsf{sign}(f_{\mathsf{tpm}}, k_I) , \mathsf{vk}(k_I)) = \alpha_1$$

Abstraction of zero-knowledge

TPM/User

Verifier

$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)}$$



statement

private messages

public messages

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(\ f_{\mathrm{tpm}}\ ,\ \mathsf{sign}(f_{\mathrm{tpm}}, k_I)\ ;\ \mathsf{vk}(k_I)\ ,\ m)$$

$$\mathsf{chk}^\#(\ \mathsf{sign}(f_{\mathrm{tpm}}, k_I)\ ,\ \mathsf{vk}(k_I)) = \quad f_{\mathrm{tpm}}$$

DAA signing protocol (simplified)

TPM/User

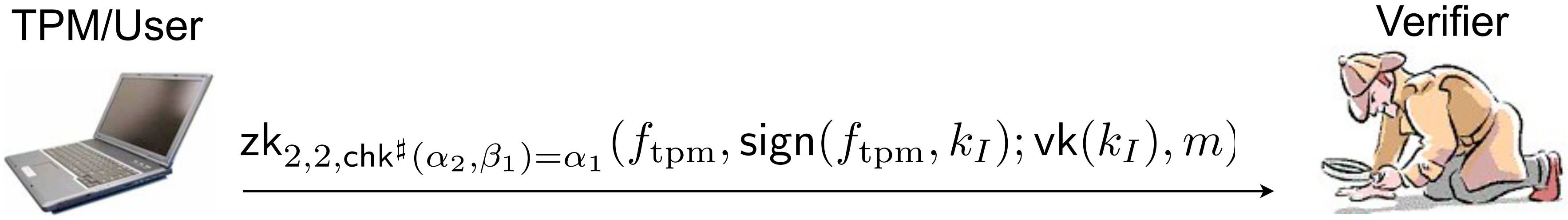
$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)}$$

Verifier

$$\begin{aligned} \mathrm{DAA} \quad = \quad & \mathsf{new}\ k_I. \\ & \mathsf{new}\ f_{\mathrm{tpm}}. \\ & \mathrm{TPM} \quad | \quad \mathrm{Verif} \quad | \quad \mathrm{Issuer} \end{aligned}$$

$$\begin{aligned} \mathrm{TPM} \quad = \quad & \mathsf{new}\ m. \\ & \mathsf{out}(c, \mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)) \end{aligned}$$

DAA signing protocol (simplified)


TPM/User
Verifier
$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$

$$\begin{array}{lcl} \mathrm{DAA} & = & \mathsf{new}\ k_I. \\ & & \mathsf{new}\ f_{\mathrm{tpm}}. \\ & & \mathrm{TPM} \quad | \quad \mathrm{Verif} \quad | \quad \mathrm{Issuer} \end{array}$$

$$\begin{array}{lcl} \mathrm{TPM} & = & \mathsf{new}\ m. \\ & & \mathsf{out}(c, \mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)) \end{array}$$

$$\begin{array}{lcl} \mathrm{Verif} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ \langle x_m \rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \end{array}$$

DAA signing protocol (simplified)

TPM/User

$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},\mathsf{sign}(f_{\mathrm{tpm}},k_I);\mathsf{vk}(k_I),m)}$$

Verifier

$$\begin{aligned}
\mathrm{DAA} \quad &= \quad \mathsf{new}\ k_I.\\
&\qquad \mathsf{new}\ f_{\mathrm{tpm}}.\\
&\qquad \mathrm{TPM} \ \mid\ \mathrm{Verif} \ \mid\ \mathrm{Issuer}\\[1em]
\mathrm{TPM} \quad &= \quad \mathsf{new}\ m.\\
&\qquad \mathsf{out}(c,\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},\mathsf{sign}(f_{\mathrm{tpm}},k_I);\mathsf{vk}(k_I),m))\\[1em]
\mathrm{Verif} \quad &= \quad \mathsf{in}(c,x).\\
&\qquad \mathsf{let}\ \langle x_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(x;\mathsf{vk}(k_I))\ \mathsf{then}
\end{aligned}$$

zero-knowledge verification

Zero-knowledge verification

$$\mathsf{ver}_{n,m,l,S}(\mathsf{zk}_{n,m,S}(\widetilde{N}; M_1, \ldots, M_m), M_1, \ldots, M_l) \Downarrow \langle M_{l+1}, \ldots, M_m \rangle$$

$$\text{iff } S\{\widetilde{N}/\widetilde{\alpha}\}\{\widetilde{M}/\widetilde{\beta}\} \Downarrow_\sharp \mathsf{true}$$

Zero-knowledge verification

$$\mathsf{ver}_{n,m,l,S}(\mathsf{zk}_{n,m,S}(\widetilde{N}; M_1,\ldots,M_m), M_1,\ldots,M_l) \Downarrow \langle M_{l+1},\ldots,M_m\rangle \quad \text{iff } S\{\widetilde{N}/\widetilde{\alpha}\}\{\widetilde{M}/\widetilde{\beta}\} \Downarrow_\sharp \mathsf{true}$$

Soundness and completeness:

Verification succeeds if and only if the proof is valid

Zero-knowledge:

Only the public messages can be extracted

For computational soundness see [Backes & Unruh, CSF 2008]

Type-checking zero-knowledge

Security annotations

$$
\begin{aligned}
\mathrm{DAA} \;&=\; \mathsf{new}\ k_I. \\
&\quad \mathsf{assume}\ \forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m)) \mid \\
&\quad \mathsf{new}\ f_{\mathrm{tpm}}. \\
&\quad \mathrm{TPM} \mid \mathrm{Verif} \mid \mathrm{Issuer}
\end{aligned}
$$

authorization policy

(OkTPM(x_f) assumed by the Issuer)

$$
\begin{aligned}
\mathrm{TPM} \;&=\; \mathsf{new}\ m. \\
&\quad \mathsf{assume}\ \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid \\
&\quad \mathsf{out}(c, \mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, (k_I)); \mathsf{vk}(k_I), m))
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Verif} \;&=\; \mathsf{in}(c, x). \\
&\quad \mathsf{let}\ \langle x_m \rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\
&\quad \mathsf{assert}\ \mathrm{Authenticate}(x_m)
\end{aligned}
$$

Security annotations

$$
\begin{aligned}
\mathrm{DAA} \;=\;& \textsf{new}\ k_I.\\
& \textsf{assume}\ \forall m.((\exists x_f.\mathrm{Send}(x_f,m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m)) \mid\\
& \textsf{new}\ f_{\mathrm{tpm}}.\\
& \mathrm{TPM} \ \mid\ \mathrm{Verif} \ \mid \mathrm{Issuer}
\end{aligned}
$$

authorization policy
(OkTPM(x_f) assumed by the Issuer)

$$
\begin{aligned}
\mathrm{TPM} \;=\;& \textsf{new}\ m.\\
& \textsf{assume}\ \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid\\
& \textsf{out}(c, \textsf{zk}_{2,2,\textsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \textsf{sign}(f_{\mathrm{tpm}}, (k_I)); \textsf{vk}(k_I), m))
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Verif} \;=\;& \textsf{in}(c, x).\\
& \textsf{let}\ \langle x_m \rangle = \textsf{ver}_{2,2,\textsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(x; \textsf{vk}(k_I))\ \textsf{then}\\
& \textsf{assert}\ \mathrm{Authenticate}(x_m)
\end{aligned}
$$

Safety

A process is *safe* if each *assertion* is entailed at run-time by the current *assumptions*

Security annotations

$\mathrm{DAA} \;=\;$ new k_I.
assume $\forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f)) \Rightarrow \mathrm{Authenticate}(m))$ |
new f_{tpm}.
$\mathrm{TPM} \mid \mathrm{Verif} \mid \mathrm{Issuer}$

authorization policy
(OkTPM(x_f) assumed by the Issuer)

$\mathrm{TPM} \;=\;$ new m.
assume $\mathrm{Send}(f_{\mathrm{tpm}}, m)$ |
out$(c, \mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, (k_I)); \mathsf{vk}(k_I), m))$

$\mathrm{Verif} \;=\;$ in(c, x).
let $\langle x_m \rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(x; \mathsf{vk}(k_I))$ then
assert $\mathrm{Authenticate}(x_m)$

Robust safety

A process is *robustly safe* if it is safe when run in parallel with an arbitrary opponent process.

Basic Types

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

$\mathrm{TPM} \quad = \quad$ new m: Un.
assume Send
out$(c, \mathsf{zk}_{2,2,}$ $f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m))$

$\mathrm{Verif} \quad = \quad$ in(c, x).
let $\langle x_m \rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \mathsf{vk}(k_I))$ then
assert $\mathrm{Authenticate}(x_m)$

Type of
messages known to
the attacker

Basic Types

new k_I.
assume $\forall m.((\exists x_f.\mathrm{Send}(x_f, m) \wedge \mathrm{OkTPM}(x_f) \Rightarrow \mathrm{Authenticate}(m))$ |
new f_{tpm}: Private.
$\mathrm{TPM} \quad | \quad \mathrm{Verif} \quad |$

Type of messages unknown to the attacker

$\mathrm{TPM} \; = \;$ new m
assume S
out$(c,$ zk$_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}},$ sign(f_{tpm}, k_I); vk$(k_I), m))$

$\mathrm{Verif} \; = \;$ in(c, x).
let $\langle x_m \rangle =$ ver$_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x;$ vk$(k_I))$ then
assert $\mathrm{Authenticate}(x_m)$

Basic Types

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

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

new f_{tpm}: $\mathsf{Private}$.

$\mathrm{TPM} \quad | \quad \mathrm{Verif} \quad | \quad \mathrm{Issuer}$

$\mathrm{TPM} \;=\;$ new m: Un.

assume $\mathrm{Send}(f$

$\mathsf{out}(c, \mathsf{zk}_{2,2,\mathsf{chk}^\sharp(}$

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

let $\langle x_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,}$

assert $\mathrm{Authenticate}(x_m)$

Refinement type

[Bengtson et al., CSF 2008]

The key is used to sign only messages x_f of type Private such that OkTPM(x_f) is entailed by the current assumptions

Basic Types

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

$$\begin{array}{rcl}
\mathrm{TPM} & = & \text{new } m\text{: Un.} \\
 & & \text{assume } \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid \\
 & & \text{out}(c, \text{zk}_{2,2,\text{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \text{sign}(f_{\mathrm{tpm}}, k_I); \text{vk}(k_I), m)) \\
\mathrm{Verif} & = & \text{in}(c, x). \\
 & & \text{let } \langle x_m \rangle = \text{ver}_{2,2,\text{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \text{vk}(k_I)) \text{ then} \\
 & & \text{assert } \mathrm{Authenticate}(x_m)
\end{array}$$

The user knows that Send(f_{tpm},m) (*local assumption*) and OkTPM(f_{tpm}) (*signature check*) are entailed...
but the verifier doesn't!

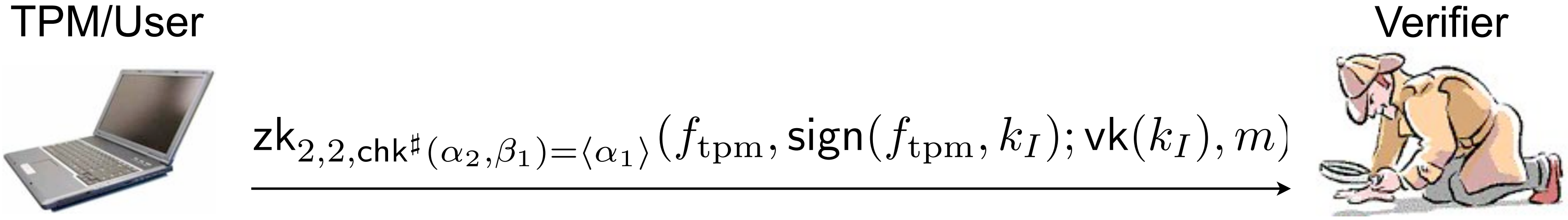
Basic Types

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

$$\begin{aligned}
\mathrm{TPM} \quad &= \quad \text{new } m\text{: Un.} \\
& \text{assume } \mathrm{Send}(f_{\mathrm{tpm}}, m) \mid \\
& \text{out}(c, \text{zk}_{2,2,\text{chk}^\sharp(\alpha_2,\beta_1)=\alpha_1}(f_{\mathrm{tpm}}, \text{sign}(f_{\mathrm{tpm}}, k_I); \text{vk}(k_I), m)) \\
\mathrm{Verif} \quad &= \quad \text{in}(c, x). \\
& \text{let } \langle x_m \rangle = \text{ver}_{2,2,\text{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \text{vk}(k_I)) \text{ then} \\
& \text{assert } \mathrm{Authenticate}(x_m)
\end{aligned}$$

How can we statically transfer these predicates from the user to the verifier?

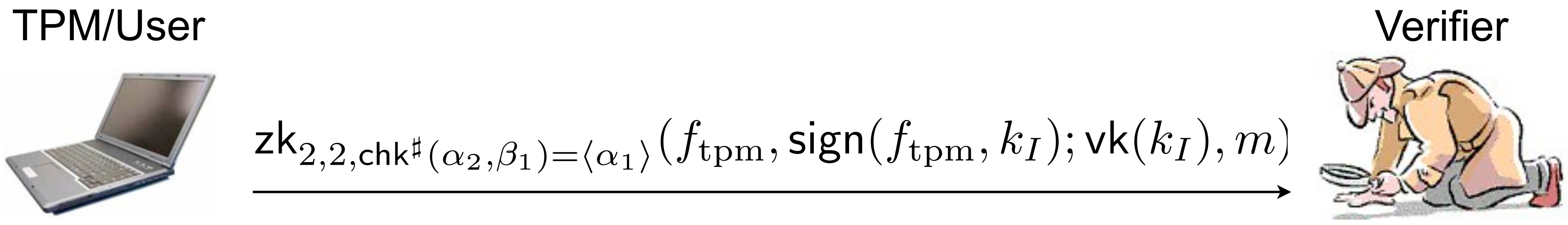
Typing zero-knowledge proofs



$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}},\mathsf{sign}(f_{\mathrm{tpm}},k_I);\mathsf{vk}(k_I),m)}$$

As usual! Use a refinement type for the key and ...

Typing zero-knowledge proofs



TPM/User

Verifier

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$$

Technical issue

Zero-knowledge proofs don't necessarily rely on keys...

Typing zero-knowledge proofs

TPM/User

$$\xrightarrow{\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}},\mathsf{sign}(f_{\mathrm{tpm}},k_I);\mathsf{vk}(k_I),m)}$$

Verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

For each statement in the protocol the user needs to annotate such a type

Typing zero-knowledge proofs

TPM/User

Verifier

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m) \longrightarrow$$

Type of
public messages

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Typing zero-knowledge proofs

TPM/User

Verifier

$$\mathsf{zk}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)$$

Type of public messages

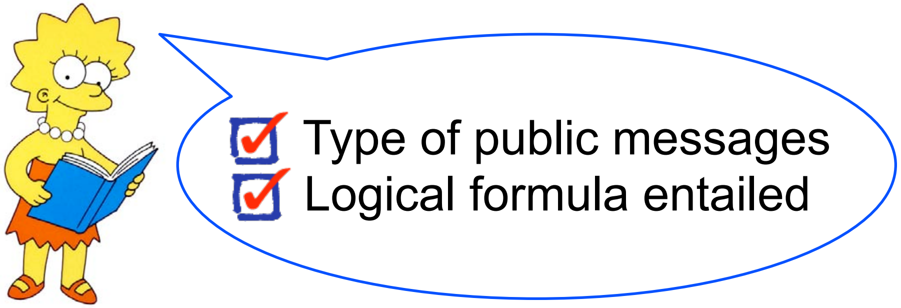
$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Formula entailed by the current assumptions (private messages existentially quantified)

Type-checking the prover

$$\mathsf{ZK}_{2,2,\mathsf{chk}^{\sharp}(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

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



☑ Type of public messages
☑ Logical formula entailed

$$\begin{array}{lcl} \mathrm{TPM} & = & \ldots \\ & & \mathsf{out}(c, \mathsf{zk}_{2,2,\mathsf{chk}^{\sharp}(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(f_{\mathrm{tpm}}, \mathsf{sign}(f_{\mathrm{tpm}}, k_I); \mathsf{vk}(k_I), m)) \end{array}$$

Type-checking the verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

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

$$\begin{aligned}\mathrm{Verif} \;=\;\; & \mathsf{in}(c, x). \\ & \mathsf{let}\ \langle y_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{aligned}$$

Type-checking the verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

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

If verification succeeds, can we give $\langle y_m\rangle$ type $\langle y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}$?

$$\begin{array}{lll}\mathrm{Verif} & = & \mathsf{in}(c, x). \\ & & \mathsf{let}\ \langle y_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \mathsf{vk}(k_I))\ \mathsf{then} \\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

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

If verification succeeds, can we give $\langle y_m\rangle$ type $\langle y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}$?

In general not!

$$\begin{array}{lcl}\mathrm{Verif} & = & \mathsf{in}(c, x).\\ & & \mathsf{let}\ \langle y_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \mathsf{vk}(k_I))\ \mathsf{then}\\ & & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{array}$$

Type-checking the verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

$\Gamma \;=\; \ldots$

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

$f_{\mathrm{tpm}} : \mathsf{Private},$

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

Does the zero-knowledge proof come from the adversary or from an honest participant?

If verification succeeds, can we give $\langle y_m\rangle$ type $\langle y_m : \mathsf{Un}\rangle\{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}$?

In general not!

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

$\mathsf{let}\ \langle y_m\rangle = \mathsf{ver}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}(x; \mathsf{vk}(k_I))\ \mathsf{then}$

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

Type-checking the verifier

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Conceptual issue

We do not know whether the zero-knowledge proof comes from an honest participant or from the adversary!

Zero-knowledge verification

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Statement

$$\mathsf{chk}^\sharp(x_s, \mathsf{vk}(k_I)) = x_f$$

Typing environment

$$\mathsf{vk}(k_I) : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\})$$

Take the statement (instantiated with the public messages you know) and the typing environment

Zero-knowledge verification

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Statement

$$\mathsf{chk}^\sharp(x_s, \mathsf{vk}(k_I)) = x_f$$

Typing environment

$$\mathsf{vk}(k_I) : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\})$$

$$\downarrow$$

$$\ldots, x_f : \mathsf{Private},$$
$$\mathrm{OkTPM}(x_f)$$

The type of the signing key gives us the type of the first private message (existentially quantified)!

Zero-knowledge verification

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle \\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \land \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Statement

$$\mathsf{chk}^\sharp(x_s, \mathsf{vk}(k_I)) = x_f$$

Typing environment

$$\mathsf{vk}(k_I) : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\})$$

$$\downarrow$$

$$\ldots, x_f : \mathsf{Private}, \mathrm{OkTPM}(x_f)$$

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

Zero-knowledge verification

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Statement

$$\mathsf{chk}^\sharp(x_s, \mathsf{vk}(k_I)) = x_f$$

Typing environment

$$\mathsf{vk}(k_I) : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\})$$

$$\downarrow$$

$$\ldots, x_f : \mathsf{Private}, \mathrm{OkTPM}(x_f)$$

$$\downarrow$$

$$\ldots, x_f : \mathsf{Private}, y_m : \mathsf{Un}, \mathrm{OkTPM}(x_f), \mathrm{Send}(x_f, y_m)$$

We can now exploit the type of the zero-knowledge proof!

Zero-knowledge verification

$$\mathsf{ZK}_{2,2,\mathsf{chk}^\sharp(\alpha_2,\beta_1)=\langle\alpha_1\rangle}\left(\begin{array}{c}\langle y_k : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\}), y_m : \mathsf{Un}\rangle\\ \{\exists x_f, x_s.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)\}\end{array}\right)$$

Statement

$\mathsf{chk}^\sharp(x_s, \mathsf{vk}(k_I)) = x_f$

Typing environment

$\mathsf{vk}(k_I) : \mathsf{VerKey}(\langle x : \mathsf{Private}\rangle\{\mathrm{OkTPM}(x)\})$

↓

$\ldots, x_f : \mathsf{Private},$
$\mathrm{OkTPM}(x_f)$

↓

$\ldots, x_f : \mathsf{Private}, y_m : \mathsf{Un}$
$\mathrm{OkTPM}(x_f), \mathrm{Send}(x_f, y_m)$

$y_m : \mathsf{Un}.$
$\exists x_f.\mathrm{Send}(x_f, y_m) \wedge \mathrm{OkTPM}(x_f)$

Type-checking the verifier

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

$$\begin{aligned}\mathrm{Verif} \;=\;\; & \ldots \\ & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{aligned}$$

Type-checking the verifier

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

$$\begin{aligned}\mathrm{Verif} \;=\;\; & \ldots \\ & \mathsf{assert}\ \mathrm{Authenticate}(y_m)\end{aligned}$$

Theorem (Robust safety)

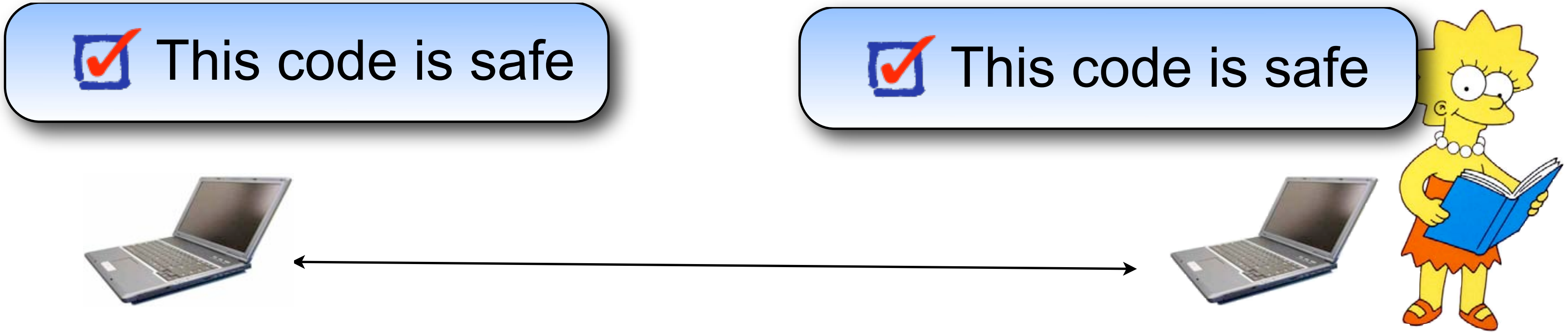
If $\Gamma \vdash$ P , then P is robustly safe

Typed analysis of zero-knowledge

- Fully automated (we implemented a type-checker and use SPASS to discharge FOL proof obligations)
- Efficient (analysis of DAA takes less than 3s)
- Compositional and therefore scalable
- Predictable termination behavior
- No explicit constraints on the semantics of destructors

Typed analysis of zero-knowledge

- Fully automated (we implemented a type-checker and use SPASS to discharge FOL proof obligations)
- Efficient (analysis of DAA takes less than 3s)
- Compositional and therefore scalable
- Predictable termination behavior
- No explicit constraints on the semantics of destructors

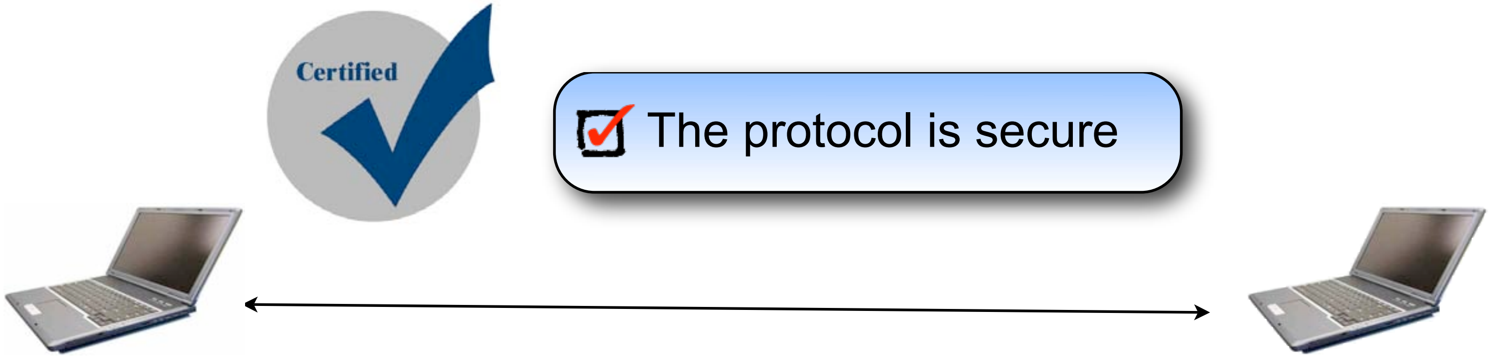


☑ This code is safe

☑ This code is safe

Typed analysis of zero-knowledge

- Fully automated (we implemented a type-checker and use SPASS to discharge FOL proof obligations)
- Efficient (analysis of DAA takes less than 3s)
- Compositional and therefore scalable
- Predictable termination behavior
- No explicit constraints on the semantics of destructors



☑ The protocol is secure

Take home

- Zero-knowledge proofs are given *refinement types* where the private messages are *existentially quantified*
- The *prover* asserts *only valid statements*
- The *verifier* can assume the formula in the type if
  - the formula is entirely derived from the zero-knowledge statement (often too weak)
  - the proof comes from an *honest party* (*statically checked* by looking at the statement and at the type of the matched public messages)

Future work

- *Sound implementation* of our abstraction
  - Identified assumptions for computational soundness in [Backes & Unruh, CSF 2008]

Future work

- *Sound implementation* of our abstraction
  - Identified assumptions for computational soundness in [Backes & Unruh, CSF 2008]
- *Type-checking implementations* of protocols using zero-knowledge
  - Extend [Bengtson et al., CSF 2008]

Future work

- *Sound implementation* of our abstraction
  - Identified assumptions for computational soundness in [Backes & Unruh, CSF 2008]
- *Type-checking implementations* of protocols using zero-knowledge
  - Extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise:*
  - Automatic protocol transformation that preserves well-typing even with corrupted participants

Future work

- *Sound implementation* of our abstraction
  - Identified assumptions for computational soundness in [Backes & Unruh, CSF 2008]
- *Type-checking implementations* of protocols using zero-knowledge
  - Extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise:*
  - Automatic protocol transformation that preserves well-typing even with corrupted participants
- Type-checking a model of *Civitas*
  - Civitas = remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]

Future work

- *Sound implementation* of our abstraction
  - Identified assumptions for computational soundness in [Backes & Unruh, CSF 2008]
- *Type-checking implementations* of protocols using zero-knowledge
  - Extend [Bengtson et al., CSF 2008]
- *Improving security despite compromise:*
  - Automatic protocol transformation that preserves well-typing even with corrupted participants
- Type-checking a model of *Civitas*
  - Civitas = remote electronic voting system [Clarkson, Chong & Myers, S&P 2008]

THANK YOU!