

# Automated Symbolic Proofs of Observational Equivalence

Jannik Dreier

LORIA, Université de Lorraine, Nancy



joint work with:

David Basin, Ralf Sasse - ETH Zurich, Institute of Information Security  
Charles Duménil, Steve Kremer - LORIA, INRIA, Nancy

Séminaire sur la Confiance Numérique  
Clermont-Ferrand  
November 3, 2016

Distributed IT systems rely on **protocols**.

They

- **specify** how the different participants interact
- provide **functionality**
- ensure – if necessary – **security** properties throughout the interaction



Distributed IT systems rely on **protocols**.

They

- **specify** how the different participants interact
- provide **functionality**
- ensure – if necessary – **security** properties throughout the interaction

Security protocol design is **critical** and **error-prone**.



Distributed IT systems rely on **protocols**.  
They

- **specify** how the different participants interact
- provide **functionality**
- ensure – if necessary – **security** properties throughout the interaction

Security protocol design is **critical** and **error-prone**.



**How to be convinced that a protocol is actually secure?**

Distributed IT systems rely on **protocols**.

They

- **specify** how the different participants interact
- provide **functionality**
- ensure – if necessary – **security** properties throughout the interaction





Security protocol design is **critical** and **error-prone**.

**How to be convinced that a protocol is actually secure?**

Use **formal methods**: **prove** the **absence of attacks** in a **formal model** under certain assumptions

# Symbolic vs. Computational Model

Two main approaches to analyze and prove protocol security:

	<b>Symbolic Model</b> 	<b>Computational Model</b> 
Messages	Terms	Bitstrings
Attacker	Dolev-Yao	Probabilistic Polynomial Time Turing Machine
Assumptions	Perfect Cryptography	Computational Hardness
Properties	Trace properties (Reachability, Correspondence) or <b>Equivalence Properties</b>	Negligible Probability of Winning the Security Game

To prove that a **protocol**  $P$  ensures a **property**  $\phi$

$$P \models \phi$$

we need a

- **formal model** with precise semantics
- **specification** of the protocol  $P$  in the formal model
- **definition** of the property  $\phi$  in the formal model

Many **tools** for protocol security analysis:

- CaserFDR [Low98]
- ProVerif [Bla01], CryptoVerif [Bla06]
- AVISPA [ABB<sup>+</sup>05]
- Scyther [Cre08], **Tamarin** [SMCB12, MSCB13, BDS15]
- CertiCrypt [BGZB09], EasyCrypt [BGHQB11]
- F7 [FKS11]
- KISS [CDK12], AKISS [CCK12]



Can be used to obtain **proofs** or **certified implementations**

- Certified Email [AB05]
- IEEE 802.11i (WiFi) [HSD<sup>+</sup>05]
- Transport Layer Security (TLS) [BFK<sup>+</sup>13]

but also to identify **attacks** and **weaknesses**



- Needham-Schroeder Protocol [Low96]
- SSL 3.0 [MSS98]
- PKCS#11 standard [DKS10]
- Helios voting system [SC11, BCP<sup>+</sup>11, BPW12]



Two types of properties:

- **Trace properties:**

- (Weak) secrecy as reachability
- Authentication as correspondence
- Defined as properties on traces
- Protocol is secure if property holds on all traces



- **Observational equivalence**

- Stronger notions of secrecy (privacy ...)
- Compares two protocol instances
- Protocol is secure if intruder cannot distinguish both instances



# Why observational equivalence?

Consider classic **Dolev-Yao** intruder for deterministic public-key encryption:

$$\frac{\text{enc}(x, pk(k)) \quad k}{x}$$

- Intruder can only decrypt if he knows the secret key

# Why observational equivalence?

Consider classic **Dolev-Yao** intruder for deterministic public-key encryption:

$$\frac{enc(x, pk(k)) \quad k}{x}$$

- Intruder can only decrypt if he knows the secret key

Now consider a simple **voting** system:

- Voter chooses  $v = \text{"Yes"}$  or  $v = \text{"No"}$
- Encrypt  $v$  using server's public key  $pk(k)$ :  $c = enc(v, pk(k))$
- Send  $c$  to server

# Why observational equivalence?

Consider classic **Dolev-Yao** intruder for deterministic public-key encryption:

$$\frac{enc(x, pk(k)) \quad k}{x}$$

- Intruder can only decrypt if he knows the secret key

Now consider a simple **voting** system:

- Voter chooses  $v = \text{"Yes"}$  or  $v = \text{"No"}$
- Encrypt  $v$  using server's public key  $pk(k)$ :  $c = enc(v, pk(k))$
- Send  $c$  to server

Is the vote **secret**?

- Dolev-Yao: **Yes**, intruder does not know server's secret key

# Why observational equivalence?

Consider classic **Dolev-Yao** intruder for deterministic public-key encryption:

$$\frac{enc(x, pk(k)) \quad k}{x}$$

- Intruder can only decrypt if he knows the secret key

Now consider a simple **voting** system:

- Voter chooses  $v = \text{"Yes"}$  or  $v = \text{"No"}$
- Encrypt  $v$  using server's public key  $pk(k)$ :  $c = enc(v, pk(k))$
- Send  $c$  to server

Is the vote **secret**?

- Dolev-Yao: **Yes**, intruder does not know server's secret key
- Reality: **No**, encryption is deterministic and there are only two choices
  - **Attack**: encrypt "Yes", and compare to  $c$

# Observational Equivalence vs Reachability

- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder cannot distinguish
  - a system where the voter votes “Yes” from
  - a system where the voter votes “No”

# Observational Equivalence vs Reachability

- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder cannot distinguish
  - a system where the voter votes “Yes” from
  - a system where the voter votes “No”
- **Observational equivalence** between two systems

# Observational Equivalence vs Reachability

- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder cannot distinguish
  - a system where the voter votes “Yes” from
  - a system where the voter votes “No”
- **Observational equivalence** between two systems
- Can be used to express:
  - Strong secrecy
  - Privacy notions, including unlinkability
  - Game-based notions, e.g., ciphertext indistinguishability



- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion

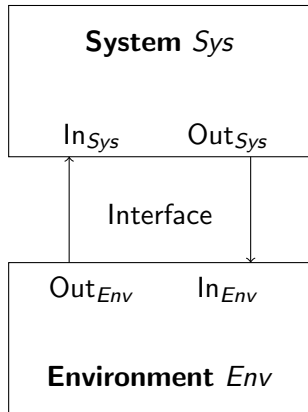
- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion

- **Auction** system
- Property: **strong secrecy** of bids

- **Auction** system
- Property: **strong secrecy** of bids
- Property **violated**: **Shout-out auction**
  - Broadcast bid (e.g., A or B)
  - Send “A” in first system
  - Send “B” in second system
  - Observer knows if he is observing first or second system

- **Auction** system
- Property: **strong secrecy** of bids
- Property **violated**: **Shout-out auction**
  - Broadcast bid (e.g., A or B)
  - Send “A” in first system
  - Send “B” in second system
  - Observer knows if he is observing first or second system
- Property **holds**: using **shared symmetric key**
  - Shared symmetric key  $k$  between bidder and auctioneer
  - Send “ $\{A\}_k$ ” in first system
  - Send “ $\{B\}_k$ ” in second system
  - Observer has no access to  $k$ , does not know which system he is observing

- We separate **environment** and **system**
  - System: agents running according to protocol
  - Environment: adversary acting according to its capabilities
- Environment can observe:
  - Output of the system
  - If system reacts at all



# Defining observational equivalence

- **Two system specifications** given as set of rules
  - One rule per role action (send/receive)
  - Running example shout-out auction:

$$\text{System 1: } \frac{}{\text{Out}_{\text{Sys}}(A)} \qquad \text{System 2: } \frac{}{\text{Out}_{\text{Sys}}(B)}$$

- Interface and environment/adversary rule(s):

$$\frac{\text{Out}_{\text{Sys}}(X)}{\text{In}_{\text{Env}}(X)} \qquad \frac{\text{Out}_{\text{Env}}(X)}{\text{In}_{\text{Sys}}(X)} \qquad \frac{\text{In}_{\text{Env}}(X) \quad K(X)}{\text{Out}_{\text{Env}}(\text{true})}$$

- $K(X)$  represents that environment knows term  $X$
- last rule models comparisons by the adversary
- Each specification yields a labeled transition system
- Observational equivalence is a kind of **bisimulation** accounting for the adversaries' **viewpoint** and **capabilities**
  - Our definition can be instantiated for various adversaries

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence**
- 4 Applications
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion



- General definition **difficult** to verify: requires inventing simulation relation
- Idea: **specialize** for cryptographic protocols
  - Consider strong bid secrecy:
    - both systems differ in **secret bid only**, i.e.
    - both specifications contain **same rule(s)** which differ only in **some terms**
  - Exploit this similarity in description and proof
- Approach: two systems described by one **specification** – using **diff**-terms

- General definition **difficult** to verify: requires inventing simulation relation
- Idea: **specialize** for cryptographic protocols
  - Consider strong bid secrecy:
    - both systems differ in **secret bid only**, i.e.
    - both specifications contain **same rule(s)** which differ only in **some terms**
  - Exploit this similarity in description and proof
- Approach: two systems described by one **specification** – using **diff**-terms
  - Running example

$$\overline{\text{Out}_{\text{Sys}}(A)}$$

$$\overline{\text{Out}_{\text{Sys}}(B)}$$

- Is equivalent to one rule with a **diff**-term

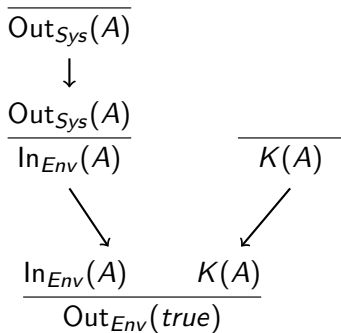
$$\overline{\text{Out}_{\text{Sys}}(\mathbf{diff}(A, B))}$$

# Approximating observational equivalence using mirroring

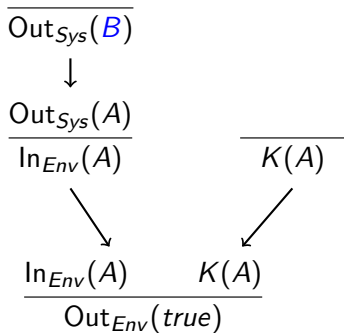
- Both systems contain the same rules modulo diff-terms
- Idea: assume that each rule simulates itself
- **Mirrors** each execution into the other system
- If the mirrors are **valid executions**, we have **observational equivalence** (sound approximation)
- We represent executions using **dependency graphs**
  - Computed via backwards constraint solving

Bidder picks A, observer compares to public value A

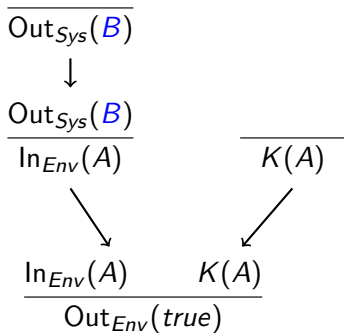
Bidder picks A, observer compares to public value A



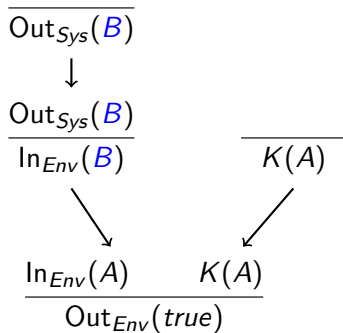
Bidder picks  $B$ , observer compares to public value  $A$



Bidder picks  $B$ , observer compares to public value  $A$

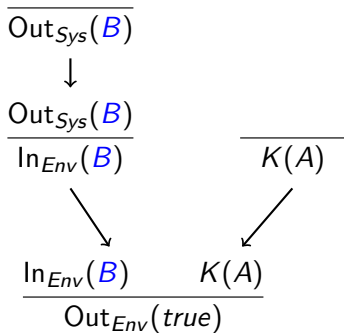


Bidder picks  $B$ , observer compares to public value  $A$

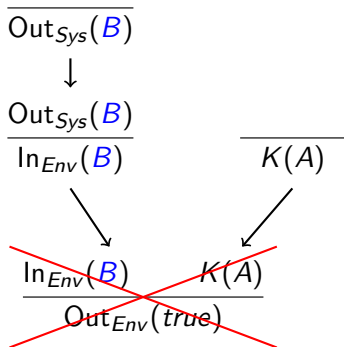




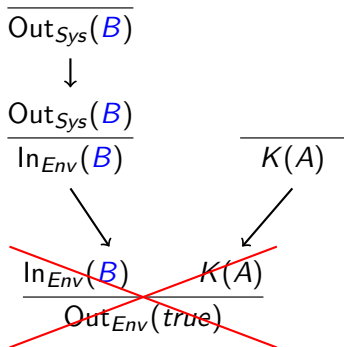
Bidder picks  $B$ , observer compares to public value  $A$



Bidder picks  $B$ , observer compares to public value  $A$



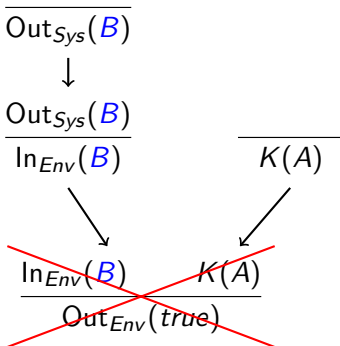
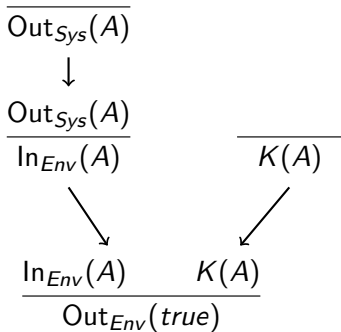
Bidder picks  $B$ , observer compares to public value  $A$



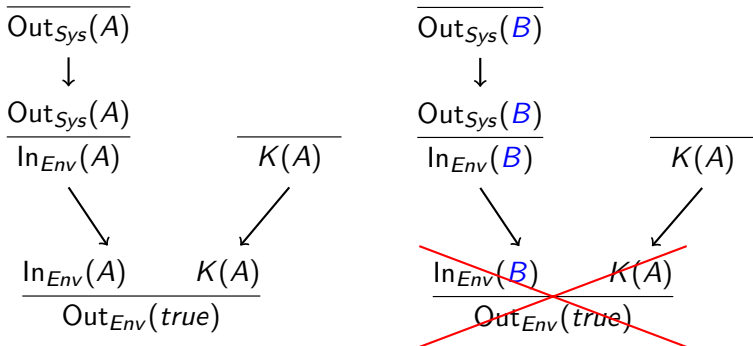
- Dependency graph mirror for bidder choice  $B$  is **invalid**
  - Adversary choices stay fixed, comparison is with  $A$

Bidder picks A/B, observer compares to public value A

Bidder picks A/B, observer compares to public value A

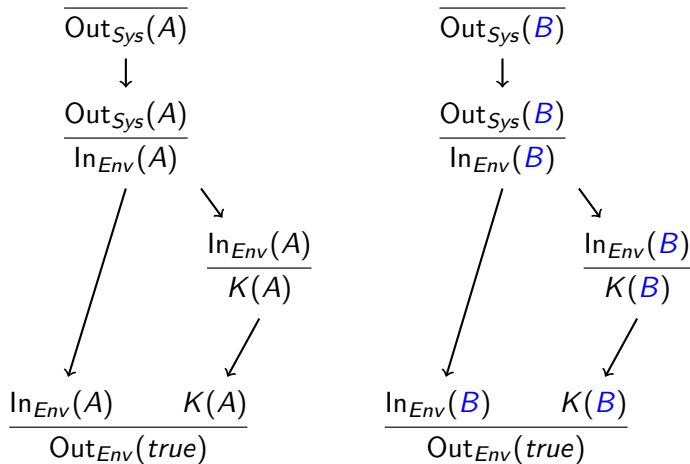


Bidder picks A/B, observer compares to public value A

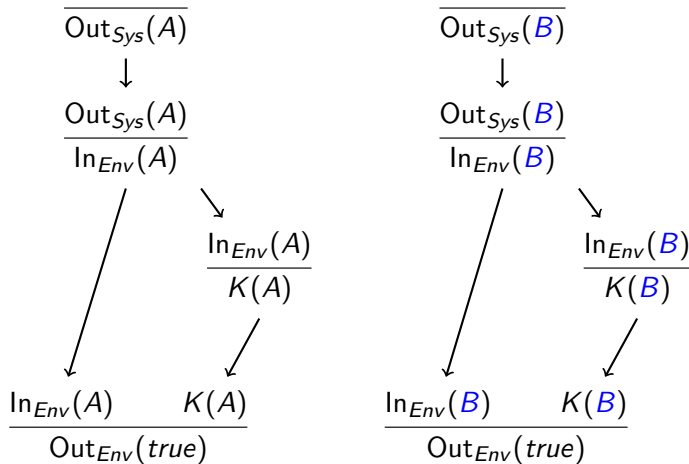


- **Counter example** to observational equivalence of the given systems

Observer compares system output to itself



Observer compares system output to itself



- **All** mirrors need to be valid for observational equivalence



A **diff**-system is **dependency graph equivalent** if mirrors of all dependency graphs rooted in any rule on both sides are valid.

- Sound but incomplete approximation
- Efficient and sufficient in practice

A **diff**-system is **dependency graph equivalent** if mirrors of all dependency graphs rooted in any rule on both sides are valid.

- Sound but incomplete approximation
- Efficient and sufficient in practice

Input:

- Protocol specification
- Property: equivalence given two choices for some term(s)
  - Example: random value vs expected value

A **diff**-system is **dependency graph equivalent** if mirrors of all dependency graphs rooted in any rule on both sides are valid.

- Sound but incomplete approximation
- Efficient and sufficient in practice

## Input:

- Protocol specification
- Property: equivalence given two choices for some term(s)
  - Example: random value vs expected value

## Output:


- **Yes**, observational equivalent
- **No**, dependency graph with invalid mirror
- Non-termination possible

Approach implemented in the Tamarin tool:

- Tamarin supports **verification** with:
  - equational theories (DH), induction, loops, mutable state
- Security protocol model is based on **rewriting**
- Restricted First-Order Logic for security properties
- Equational theories modeling **algebraic properties** of cryptographic primitives
- Constraint-solving algorithm for analysis of **unbounded number of sessions**
- Performance good despite undecidability
- Interactive and fully automatic modes
- Parallelized for multi-core performance

# Verifying observational equivalence in Tamarin

Implemented algorithm:

- Extended constraint solving
- (**Normal**) dependency graphs
  - Important for state space reduction and termination
- Equivalence of dependency graphs by mirroring
-  Convergent equational theories to deal with blind signatures, trapdoor commitments and other complex primitives

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications**
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion

# (Electronic) Cash





# (Electronic) Cash



**Electronic Cash = digital equivalent**

# (E-)Cash: Players and Phases

Bank



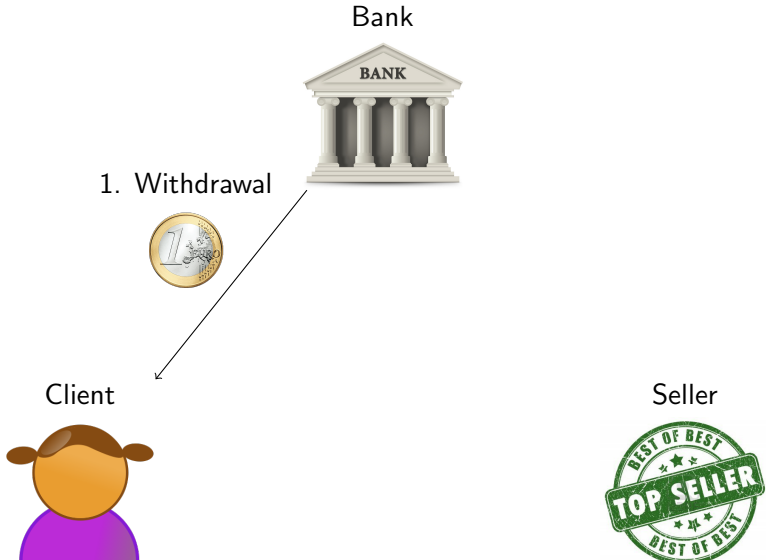
Client



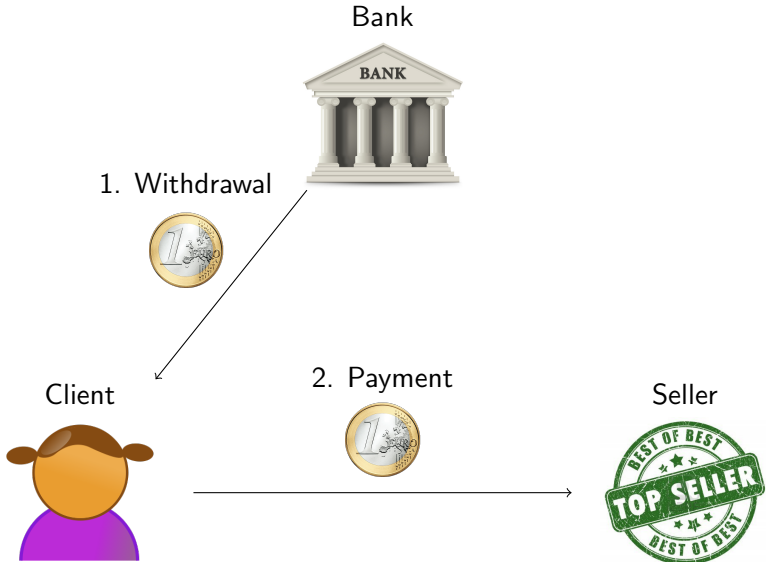
Seller



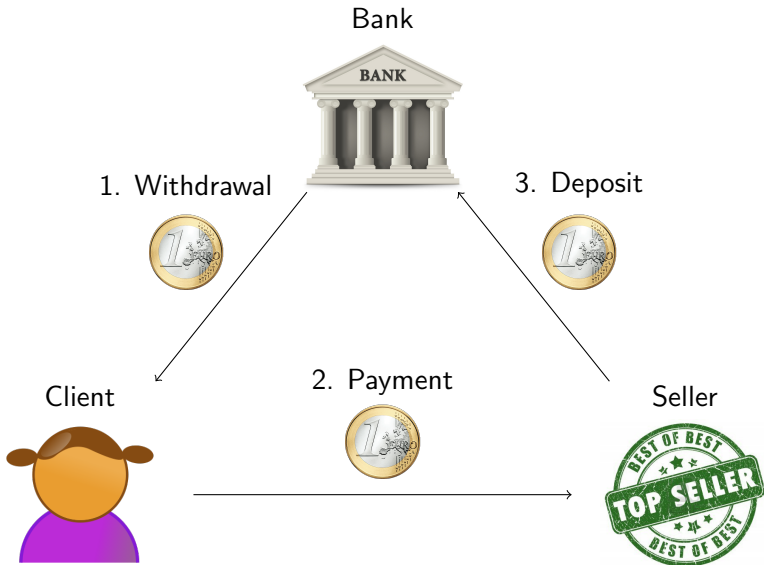
# (E-)Cash: Players and Phases



# (E-)Cash: Players and Phases



# (E-)Cash: Players and Phases



# Security properties of physical cash

- **Unforgeability:** Only the bank can create coins (trace property).
- **Anonymity:** Nobody can distinguish which client makes a payment (equivalence property).
- **Untraceability:** Nobody is able to decide whether two payments were made by the same client (equivalence property).

# Security properties of physical cash

- **Unforgeability:** Only the bank can create coins (trace property).
- **Anonymity:** Nobody can distinguish which client makes a payment (equivalence property).
- **Untraceability:** Nobody is able to decide whether two payments were made by the same client (equivalence property).
  
- **Do they really hold?**



# Electronic Cash vs. Electronic Payments



Google wallet





# Electronic Cash vs. Electronic Payments

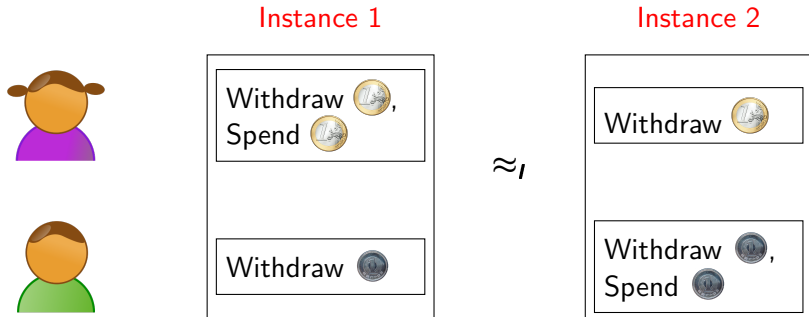


⇒ **No anonymity and unlinkability!**

Nobody can distinguish which client makes a payment.

## Definition:

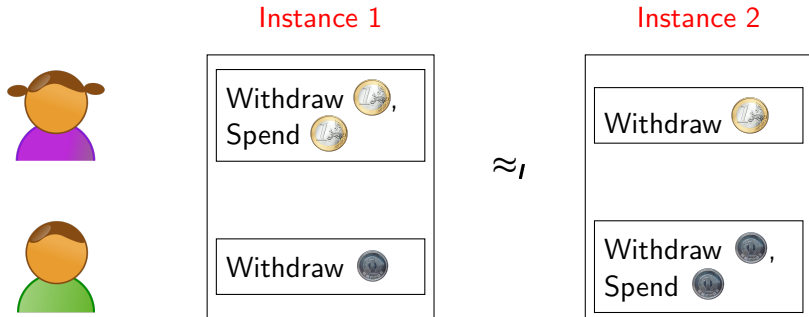
Observational equivalence of two instances:



Nobody can distinguish which client makes a payment.

**Definition:**

Observational equivalence of two instances:



Note that the **bank** and the **seller** are **corrupted**.

Nobody is able to decide whether two payments were made by the same client:



$\approx$



**First on-line E-Cash** protocol [Cha82] using

- blind signatures:  
 $\text{unblind}(\text{sign}(\text{blind}(x, r), k), r) = \text{sign}(x, k)$
- on-line verification by the bank to prevent double spending

**Goal:** ensure


- unforgeability
- anonymity
- unlinkability

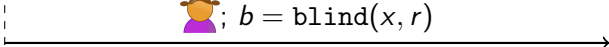
in presence of **dishonest**

- banks
- sellers
- clients

# Withdrawal Phase




;  $b = \text{blind}(x, r)$



# Withdrawal Phase



;  $b = \text{blind}(x, r)$


$s = \text{sign}(b, sk_{\text{BANK}})$

# Withdrawal Phase



$b = \text{blind}(x, r)$


$s = \text{sign}(b, sk_{\text{BANK}})$

1. Verify signature  $s$
2. Compute  $y = \text{unblind}(s, r) = \text{sign}(x, sk_{\text{BANK}})$
3. Coin   $= (x, y) = (x, \text{sign}(x, sk_{\text{BANK}}))$



# Payment and Deposit Phase




 =  $(x, \text{sign}(x, sk_{\text{bank}}))$



# Payment and Deposit Phase



 =  $(x, \text{sign}(x, sk_{\text{bank}}))$




Verify signature




# Payment and Deposit Phase



 =  $(x, \text{sign}(x, sk_{\text{bank}}))$




Verify signature

 =  $(x, \text{sign}(x, sk_{\text{bank}}))$




# Payment and Deposit Phase



 =  $(x, \text{sign}(x, sk_{\text{bank}}))$


Verify signature

 =  $(x, \text{sign}(x, sk_{\text{bank}}))$


1. Verify signature
2. Check if deposited

# Payment and Deposit Phase



 =  $(x, \text{sign}(x, sk_{\text{bank}}))$

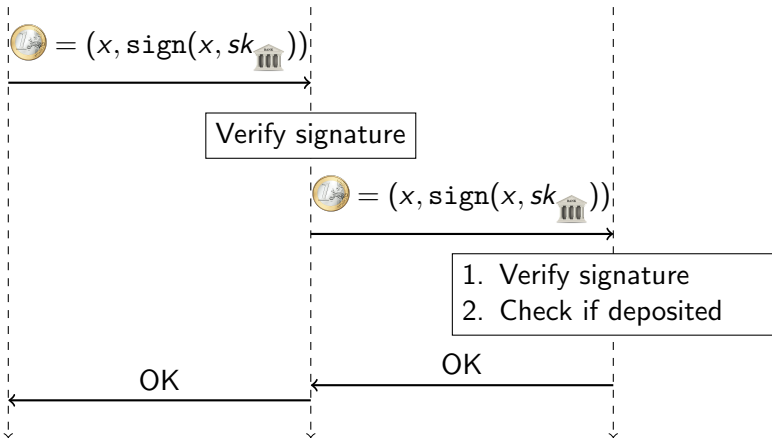
Verify signature

 =  $(x, \text{sign}(x, sk_{\text{bank}}))$

1. Verify signature
2. Check if deposited

OK

# Payment and Deposit Phase



Formal Verification with Tamarin:

Property	Result	Time
Unforgeability	✓	< 1 s
Weak Anonymity	✓	7.6 s
Strong Anonymity	✓	1 m 13.7 s

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications**
  - Chaum's e-cash protocol
  - FOO e-voting protocol**
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion



# Protocol by Fujioka, Okamoto and Ohta [FOO92]

The protocol uses

- blind signatures:

$$\text{unblind}(\text{sign}(\text{blind}(x, r), k), r) = \text{sign}(x, k)$$

- **commitments**:  $\text{open}(\text{commit}(v, r), r) = v$

to **ensure**:

- eligibility (trace property)
- vote privacy (equivalence property)

It runs in three **phases**:

- Eligibility Check
- Voting
- Counting

**Authorities**:

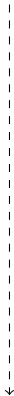
- Administrator
- Collector

**Assumptions**:

- Anonymous channel to the collector

# Eligibility Check

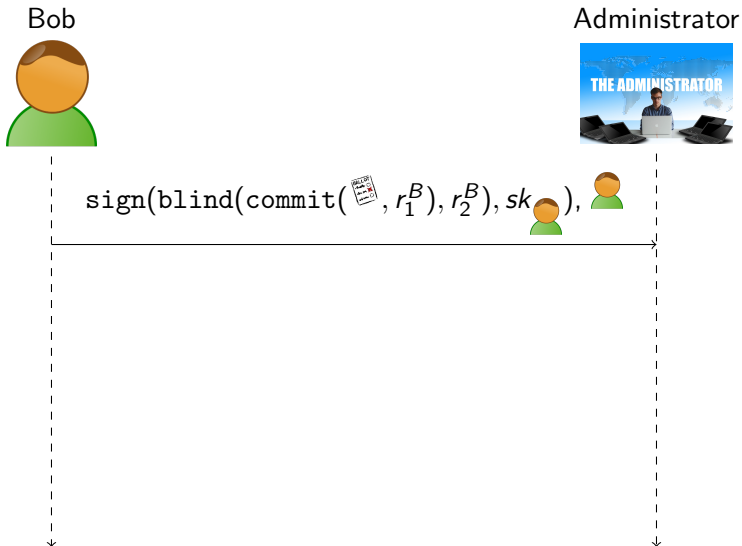
Bob



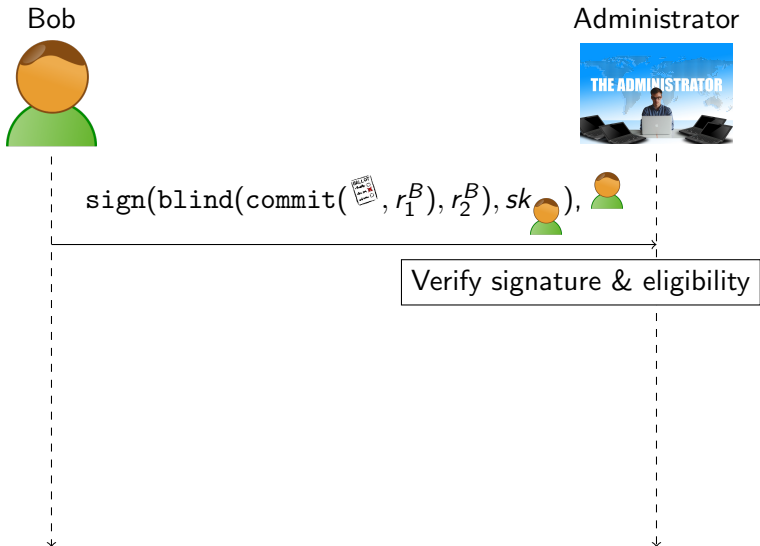
Administrator



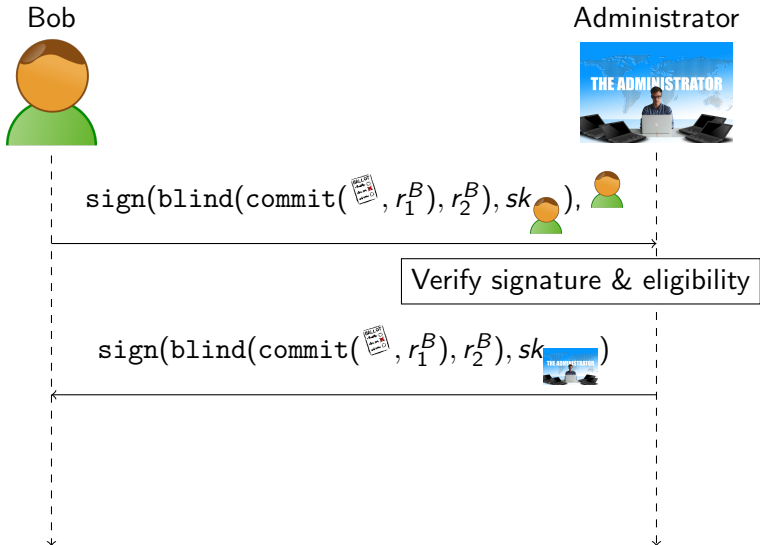
# Eligibility Check



# Eligibility Check



# Eligibility Check



# Eligibility Check

Bob



Administrator



$\text{sign}(\text{blind}(\text{commit}(\text{POLICY}, r_1^B), r_2^B), sk_{\text{Bob}}), \text{Bob}$

Verify signature & eligibility

$\text{sign}(\text{blind}(\text{commit}(\text{POLICY}, r_1^B), r_2^B), sk_{\text{Administrator}})$

Verify signature and unblind:

$\text{sign}(\text{commit}(\text{POLICY}, r_1^B), sk_{\text{Administrator}})$

Alice

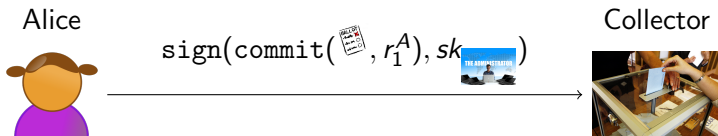


Collector



Bob

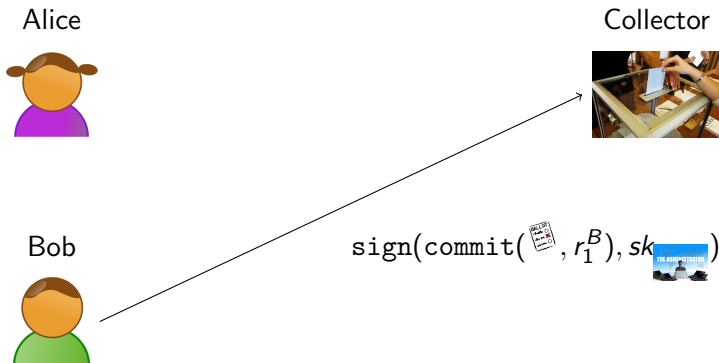




Bob







Alice





Bob

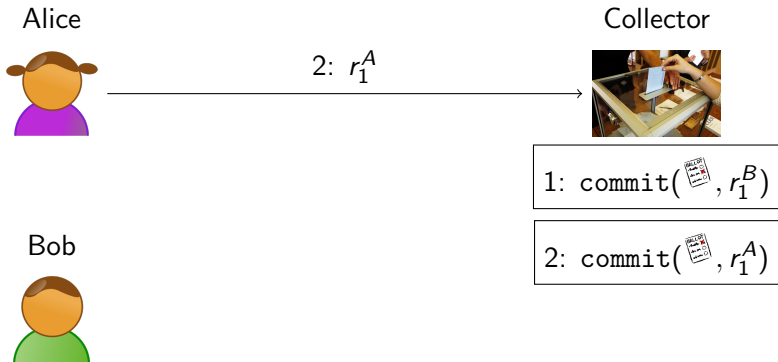


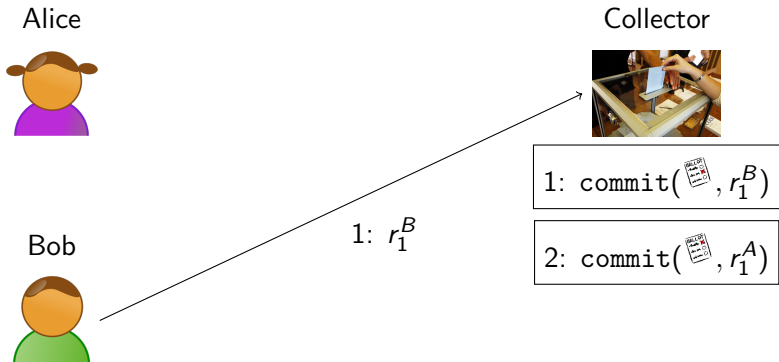
Collector



1: `commit(, r_1^B)`

2: `commit(, r_1^A)`





Alice





Bob



Collector

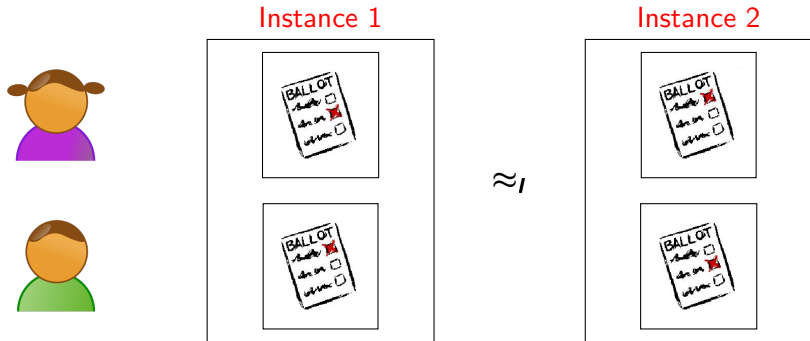


1: `commit`(,  $r_1^B$ )

2: `commit`(,  $r_1^A$ )



We define Vote-Privacy using observational equivalence between two situations:



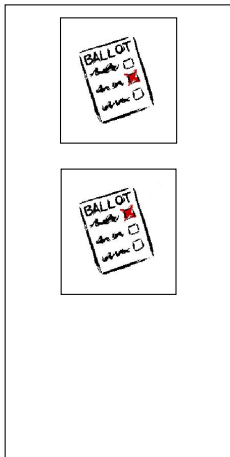
**Analysis in Tamarin:** FOO ensures Vote-Privacy and Eligibility.

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications**
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol**
  - Other examples
- 5 Conclusion

A protocol is Receipt-Free if a voter cannot construct a convincing receipt that he voted for a certain candidate.

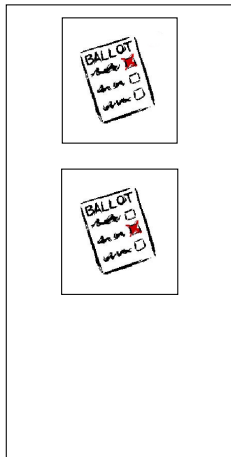


Instance 1



$\approx$

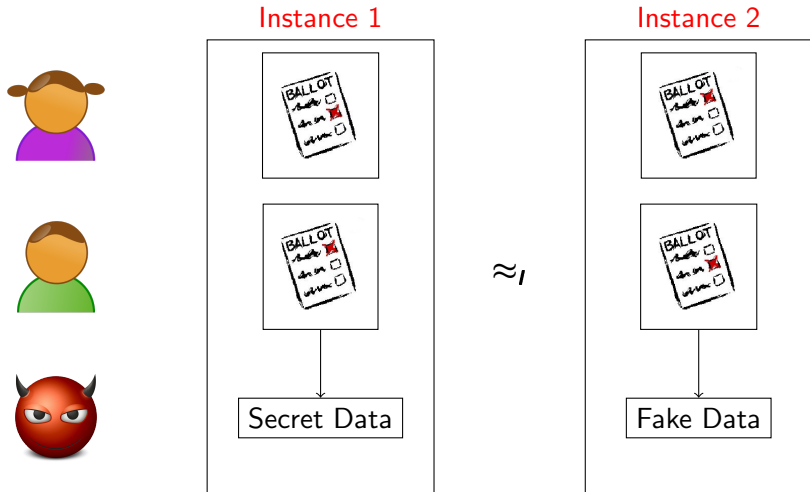
Instance 2





# Receipt-Freeness

A protocol is Receipt-Free if a voter cannot construct a convincing receipt that he voted for a certain candidate.



FOO is not **receipt-free**: a voter can prove that he voted for a certain candidate by revealing his key and his random values.

The protocol by Okamoto (an extension of FOO) addresses this using **trapdoor-commitments** which can be opened differently using a trapdoor:

- $open(tdcommit(m, r, td), r) = m$
- $open(tdcommit(m_1, r, td), f(m_1, r, td, m_2)) = m_2$
- $tdcommit(m_2, f(m_1, r, td, m_2), td) = tdcommit(m_1, r, td)$
- $f(m_1, f(m, r, td, m_1), td, m_2) = f(m, r, td, m_2)$

**Analysis in Tamarin:** the protocol by Okamoto ensures Eligibility, Vote-Privacy and Receipt-Freeness.

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications**
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples**
- 5 Conclusion

- **Signed Diffie-Hellman** key exchange
  - Special equational theory to model Diffie-Hellman exponentiation
  - Real-or-random secrecy of session key
  - Needs manual guidance in one subcase
  - Automatically completed proof in 2.5 minutes
- **TPM\_Envelope**
  - Real-or-random secrecy
  - Finds attack for deterministic encryption
    - Despite previous proof wrt trace-based secrecy
  - We recommend to use probabilistic encryption

## Summary of case studies

<b>Protocol</b>	<b>Property</b>	<b>Result</b>	<b>Time</b>
Chaum	Unforgeability	Verified	0.2s
Chaum	Anonymity	Verified	7.6s
Chaum	Untraceability	Verified	1m13.7s
FOO	Eligibility	Verified	10.3s
FOO	Vote Privacy	Verified	4m11.1s
Okamoto	Eligibility	Verified	8.4s
Okamoto	Vote Privacy	Verified	1m20.3s
Okamoto	Receipt-Freeness	Verified	13m35.8s
Signed DH Key Exchange	RoR secrecy	Verified	manual
TPM_Envelope	RoR secrecy	Attack	1.5 s

- 1 Introduction
- 2 Defining Observational Equivalence
- 3 Verifying Observational Equivalence
- 4 Applications
  - Chaum's e-cash protocol
  - FOO e-voting protocol
  - Okamoto's e-voting protocol
  - Other examples
- 5 Conclusion

- **Equivalence properties** are necessary to specify complex security properties
- Use **sound approximation** as equivalence is difficult to verify
- Resulting implementation in **Tamarin** is effective and efficient, illustrated by many **case studies**:
  - Chaum's e-cash protocol: Anonymity and Unlinkability
  - FOO e-voting protocol: Vote-Privacy
  - Okamoto e-voting protocol: Receipt-Freeness
  - Other examples: real-or-random key secrecy for Signed Diffie-Hellman, TPM\_Envelope protocol, ...
- **High** degree of **automation**
- **Future work**:
  - Implement **more precise approximation**
  - Protocols with loops: need **invariants and induction**
  - Further case studies
    - Signed Diffie-Hellman with Perfect Forward Secrecy
    - NAXOS, authenticated key exchange with PFS

Thank you for your attention!

**Questions?**

`jannik.dreier@loria.fr`

**Tamarin tool and all case studies** available at:

`http://tamarin-prover.github.io/`





Martín Abadi and Bruno Blanchet.

Computer-Assisted Verification of a Protocol for Certified Email.

*Science of Computer Programming*, 58(1–2):3–27, October 2005.

Special issue SAS'03.



Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron.

The avispa tool for the automated validation of internet security protocols and applications.

In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, pages 281–285, Berlin, Heidelberg, 2005. Springer-Verlag.



David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi.

Adapting helios for provable ballot privacy.

In *Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS'11)*, volume 6879 of LNCS, pages 335–354, Leuven, Belgium, 2011. Springer.



David Basin, Jannik Dreier, and Ralf Sasse.

Automated Symbolic Proofs of Observational Equivalence.

In *22nd ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2015)*, pages 1144–1155, Denver, United States, October 2015. ACM.



Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub.

Implementing tls with verified cryptographic security.

In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'13)*, pages 445–459, 2013.



Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin.

Computer-aided security proofs for the working cryptographer.  
In *Proceedings of the 31st Annual Cryptology Conference (CRYPTO'11)*, volume 6841 of *LNCS*, pages 71–90, Santa Barbara, CA, USA, 2011. Springer.



Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin.

Formal certification of code-based cryptographic proofs.  
In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'09)*, pages 90–101. ACM, 2009.



Bruno Blanchet.

An Efficient Cryptographic Protocol Verifier Based on Prolog Rules.

In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.



Bruno Blanchet.

A computationally sound mechanized prover for security protocols.

In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'06)*, pages 140–154, Oakland, California, May 2006.



David Bernhard, Olivier Pereira, and Bogdan Warinschi.

How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios.

In *Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'12)*, volume 7658 of LNCS, pages 626–643, Beijing, China, 2012. Springer.



Rohit Chadha, Ștefan Ciobâcă, and Steve Kremer.

Automated verification of equivalence properties of cryptographic protocols.

In *Proceedings of the 21st European Symposium on Programming Languages and Systems (ESOP'12)*, volume

7211 of LNCS, pages 108–127, Tallinn, Estonia, 2012. Springer.



Ștefan Ciobâcă, Stéphanie Delaune, and Steve Kremer.

Computing knowledge in security protocols under convergent equational theories.

*Journal of Automated Reasoning*, 48(2):219–262, 2012.



David Chaum.

Blind signatures for untraceable payments.

In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1982.



Cas J. F. Cremers.

The Scyther Tool: Verification, falsification, and analysis of security protocols.

In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of LNCS, pages 414–418, Princeton, USA, 2008. Springer.



Stéphanie Delaune, Steve Kremer, and Graham Steel.

Formal security analysis of pkcs#11 and proprietary extensions.

*Journal of Computer Security*, 18(6):1211–1245, 2010.



Cédric Fournet, Markulf Kohlweiss, and Pierre-Yves Strub.  
Modular code-based cryptographic verification.

In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS'11)*, pages 341–350, New York, NY, USA, 2011. ACM.



Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta.

A practical secret voting scheme for large scale elections.

In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques (AUSCRYPT'92)*, volume 718 of *LNCS*, pages 244–251. Springer Berlin / Heidelberg, Gold Coast, Queensland, Australia, 1992.



Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell.

A modular correctness proof of iee 802.11i and tls.

In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 2–15, Alexandria, VA, USA, 2005. ACM.



Gavin Lowe.

Breaking and fixing the needham-schroeder public-key protocol using *fd*.

In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*, pages 147–166, London, UK, 1996.

Springer-Verlag.



Gavin Lowe.

Casper: A compiler for the analysis of security protocols.  
*Journal of Computer Security*, 6(1-2):53–84, 1998.



Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin.

The tamarin prover for the symbolic analysis of security protocols.

In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701, Saint Petersburg, Russia, 2013. Springer.



John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern.  
Finite-state analysis of ssl 3.0.

In *Proceedings of the 7th USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 1998. USENIX Association.



Tatsuaki Okamoto.

An electronic voting scheme.

In *Proceedings of the IFIP World Conference on IT Tools*, pages 21–30, 1996.



Ben Smyth and Veronique Cortier.

Attacking and fixing helios: An analysis of ballot secrecy.

In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 297–311. IEEE, 2011.



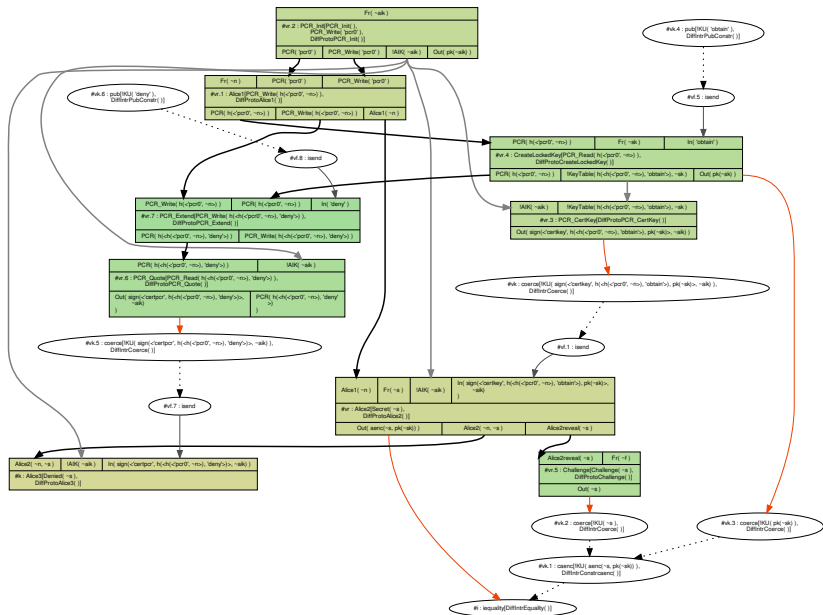


Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin.

Automated analysis of diffie-hellman protocols and advanced security properties.

In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 78–94, Cambridge, MA, USA, 2012. IEEE.

# TPM\_Envelope attack



	APTE	AKISS	ProVerif	ProVerifDH	SPEC	Maude-NPA	Tamarin	Extension
Unbounded sessions			x	x		x	x	x
Mutable state	x	x			x	?	x	x
Diffie-Hellman	x	x		x	x	x	x	x
Definable crypto	x	x	x	x		x	x	x
Verification	x	x	x	x	x	x	x	x
Obs. equiv.	x	x	x		x	/		x

- APTE, AKISS
  - Limited to bounded number of sessions
- ProVerif
  - No mutable state support
  - DH support only without observational equivalence
- SPEC
  - Fixed crypto primitives, bounded number of sessions
- StatVerif, SAPIC
  - Support mutable state, but no observational equivalence
- Maude-NPA
  - Creates synchronous product of two similar protocols
  - Suffers from termination issues - only finds attacks

# Observational equivalence - definition

Two sets of multiset rewrite rules  $S_A$  and  $S_B$  are **observational equivalent** with respect to an environment  $Env$  (and interface  $IF$ ) if there is a **relation** between the initial states in  $S_A \cup IF \cup Env$  (**left system**) and  $S_B \cup IF \cup Env$  (**right system**), and for all pairs of states in that relation:

- If the **left system** can make a **move** with an environment or interface rule, the **right system** can **match** it **precisely**
  - Resulting states are in the relation
- If the **left system** can make a **move** with an  $S_A$  rule, the **right system** can **match** it, possibly using **multiple steps**
  - resulting states are in the relation

The same holds in the other direction.

```
1: function VERIFY( $S$ )
2:    $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$ 
3:   while  $RU \neq \emptyset$  do
4:     choose  $r \in RU$ ,  $RU \leftarrow (RU \setminus \{r\})$ 
5:     compute  $DG \leftarrow dgraphs(r)$  by constraint solving
6:     if  $\exists dg \in DG$  s.t.  $mirrors(dg)$  lacks ground instances
7:       then return "potential attack found: ",  $dg$ 
8:   return "verification successful"
```

Only the bank can create coins.

## Definition:

On every trace:

