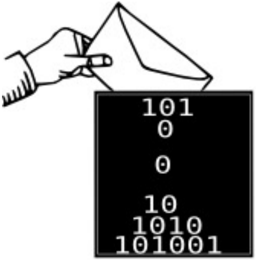
Computer Aided Security: Cryptographic Primitives, Voting protocols, and Wireless Sensor Networks

Pascal Lafourcade

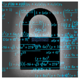
November 6, 2012
Habilitation à Diriger des Recherches

Nowadays Security is Everywhere!



What is cryptography based security?

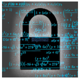
Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

What is cryptography based security?

Cryptography:



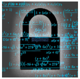
- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

- Secrecy,
- Authentication,
- Privacy ...

What is cryptography based security?

Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

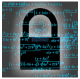
- Secrecy,
- Authentication,
- Privacy ...

Intruders:

- Passive
- Active
- CPA, CCA ...

What is cryptography based security?

Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

- Secrecy,
- Authentication,
- Privacy ...

Intruders:

- Passive
- Active
- CPA, CCA ...

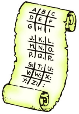
Designing **secure** cryptographic protocols is **difficult**

Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?

Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?

Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?

- Prove that there is no attack under some assumptions.

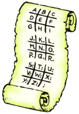
Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?

- Prove that there is no attack under some assumptions.
  - proving is a difficult task,
  - pencil-and-paper proofs are error-prone.

Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?



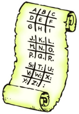


- Prove that there is no attack under some assumptions.
  - proving is a difficult task,
  - pencil-and-paper proofs are error-prone.

How can we be convinced that a proof is correct?

Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?





- Prove that there is no attack under some assumptions.
  - proving is a difficult task,
  - pencil-and-paper proofs are error-prone.

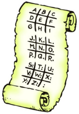
How can we be convinced that a proof is correct?





Security of Cryptographic Protocols

How can we be convinced that a protocols is secure?



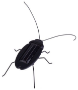


- Prove that there is no attack under some assumptions.
  - proving is a difficult task,
  - pencil-and-paper proofs are error-prone.

How can we be convinced that a proof is correct?

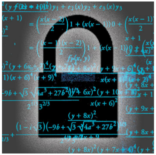






Computer-Aided Security.

Formal Verification Approaches

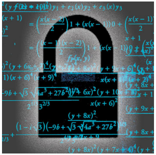


Designer

Attacker

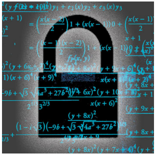
Formal Verification Approaches

Designer



Attacker

Formal Verification Approaches

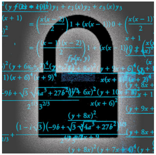


Designer

Attacker

Give a proof

Formal Verification Approaches



Designer

Attacker

Give a proof

Find a flaw

Back to 1995

     ≥ 17  (FDR)

Back to 1995

     ≥ 17  (FDR)

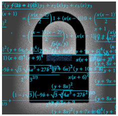
- **Cryptography:** Perfect Encryption hypothesis
- **Property:** Secrecy, Authentication
- **Intruder:**
  - Active
  - Controlling the network
  - Several sessions

Success Story of Symbolic Verification

Tools based on different theories for several properties

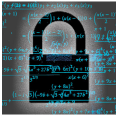
1995	Casper/FRD [Lowe]
2001	Proverif [Blanchet]
2003	Proof of certified email protocol with Proverif [AB]
	OFMC [BMV]
	Hermes [BLP]
	Flaw in Kerberos 5.0 with MSR 3.0 [BCJS]
2004	TA4SP [BHKO]
2005	SATMC [AC]
2006	CL-ATSE [Turuani]
2008	Scyther [Cremers]
	Flaw of Single Sign-On for Google Apps with SAT-MC [ACCCT]
	Proof of TLS using Proverif [BFCZ]
2010	TOOKAN [DDS] using SAT-MC for API
2012	Tamarin [BCM]

Main Contributions:



- Verification techniques for **cryptography**
  - Asymmetric Encryptions
  - Encryption Modes
  - Message Authentication Codes

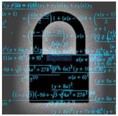
Main Contributions:



- Verification techniques for **cryptography**
  - Asymmetric Encryptions
  - Encryption Modes
  - Message Authentication Codes

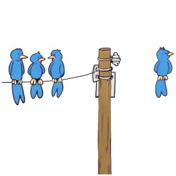
- **Properties** for E-voting protocols
  - Taxonomy of privacy notions
  - Weighted votes

Main Contributions:



- Verification techniques for **cryptography**
  - Asymmetric Encryptions
  - Encryption Modes
  - Message Authentication Codes

- **Properties** for E-voting protocols
  - Taxonomy of privacy notions
  - Weighted votes



- **Intruder** models and algorithms for WSN
  - Neighbourhood Discovery Protocols
  - Independent Intruders
  - Routing Algorithms

Outline

Related Work

- **CryptoVerif [BP06]:**
  - tool that generates proofs by sequences of games
  - has automatic and manual modes
- **CIL [BDKL10]:** Computational Indistinguishability Logic for proving cryptographic primitives.
- **CertiCrypt [BGZB09] /EasyCrypt [BGHB11]:**
  - Framework for machine-checked cryptographic proofs in Coq
  - Improved by EasyCrypt: generates CertiCrypt proofs from proof sketches

Related Work

- **CryptoVerif [BP06]:**
  - tool that generates proofs by sequences of games
  - has automatic and manual modes
- **CIL [BDKL10]:** Computational Indistinguishability Logic for proving cryptographic primitives.
- **CertiCrypt [BGZB09] /EasyCrypt [BGHB11]:**
  - Framework for machine-checked cryptographic proofs in Coq
  - Improved by EasyCrypt: generates CertiCrypt proofs from proof sketches

Our Approach

Automatically proving security of cryptographic primitives

1. Defining a language
2. Modeling security properties
3. Building a Hoare Logic for proving the security

Our Approach

Automatically proving security of cryptographic primitives

1. Defining a language
2. Modeling security properties
3. Building a Hoare Logic for proving the security

Correct but not complete.

Our Approach

Automatically proving security of cryptographic primitives

1. Defining a language
2. Modeling security properties
3. Building a Hoare Logic for proving the security

Correct but not complete.

- Asymmetric Encryption Schemes [CDELL'08,CDELL'10]
- Encryption Modes [GLLS'09]
- Message Authentication Codes (MACs) Submitted [GLL'13]

Our Approach

Automatically proving security of cryptographic primitives

1. Defining a language
2. Modeling security properties
3. Building a Hoare Logic for proving the security

Correct but not complete.

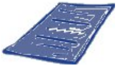
- Asymmetric Encryption Schemes [CDELL'08,CDELL'10]
- Encryption Modes [GLLS'09]
- Message Authentication Codes (MACs) Submitted [GLL'13]

Examples of Asymmetric Encryptions

- **[BR'93]**: $f(r)||x \oplus G(r)||H(x||r)$
- **[SZ'93]**: $f(r)||G(r) \oplus (x||H(x))$
- **[BR'94] OAEP**: $f(s||r \oplus H(s))$
  where $s = x0^k \oplus G(r)$
- **[Shoup'02] OAEP+**: $f(s||r \oplus H(s))$
  where $s = x \oplus G(r)||H'(r||x)$.
- **[FO'99]**: $\mathcal{E}((x||r); H(x||r))$
  where $\mathcal{E}$ is IND-CPA.

f is a one-way trapdoor permutation, H and G are hash functions and r is a random seed.

Security Property: Indistinguishability



Security Property: Indistinguishability

Security Property: Indistinguishability

Indis(x; V_1; V_2): seeing V_1 and $f(V_2)$.

Modelling: Generic Encryption Scheme

Grammar for Generic Encryption

$$\begin{array}{lcl} \mathsf{cmd} & ::= & x \leftarrow \mathcal{U} \mid x := f(y) \mid x := H(y) \mid \\ & & x := y \oplus z \mid x := y || z \mid \mathsf{cmd}; \mathsf{cmd} \end{array}$$

Modelling: Generic Encryption Scheme

Grammar for Generic Encryption

$$\begin{array}{lcl} \mathsf{cmd} & ::= & x \leftarrow \mathcal{U} \mid x := f(y) \mid x := H(y) \mid \\ & & x := y \oplus z \mid x := y || z \mid \mathsf{cmd}; \mathsf{cmd} \end{array}$$

A Generic Encryption Scheme

$$\begin{array}{l} \mathcal{E}(\mathsf{in}_e, \mathsf{out}_e) = \\ \quad c_1; \\ \quad c_2; \\ \quad \vdots \\ \quad c_n; \end{array}$$

Modelling: Generic Encryption Scheme

Grammar for Generic Encryption

$$\begin{array}{lll} \mathrm{cmd} & ::= & x \leftarrow \mathcal{U} \mid x := f(y) \mid x := H(y) \mid \\ & & x := y \oplus z \mid x := y || z \mid \mathrm{cmd}; \mathrm{cmd} \end{array}$$

A Generic Encryption Scheme

$$\begin{array}{c} \mathcal{E}(\mathrm{in}_e, \mathrm{out}_e) = \\ c_1; \\ c_2; \\ \vdots \\ c_n; \end{array}$$

Bellare & Rogaway'93:

$f(r)||\mathrm{in}_e \oplus G(r)||H(\mathrm{in}_e||r)$

$\mathcal{E}_{BR93}(\mathrm{in}_e, \mathrm{out}_e) =$
$r \xleftarrow{r} \mathcal{U};$
$a := f(r);$
$g := G(r);$
$b := \mathrm{in}_e \oplus g;$
$t := \mathrm{in}_e||r;$
$c := H(t);$
$\mathrm{out}_e := a||b||c$

Only Three Predicates in the ROM

Predicates

$$\begin{array}{lcl} \psi & ::= & \mathsf{H}(G, e) \mid \mathsf{WS}(x; V) \mid \mathsf{Indis}(x; V_1; V_2) \\ \varphi & ::= & \mathsf{true} \mid \psi \mid \varphi \wedge \varphi \end{array}$$

- **H**(G, e): Not-Hashed-Yet

  $\Pr[S \xleftarrow{r} X : S(e) \in S(\mathcal{T}_H).dom]$ is negligible.

- **WS**$(x; V)$: cannot to compute some "hidden" value.

  $\Pr[S \xleftarrow{r} X : A(S) = S(x)]$ is negligible.

- **Indis**$(x; V_1; V_2)$: seeing V_1 and $f(V_2)$.

Only Three Predicates in the ROM

Predicates

$$\begin{array}{rcl} \psi & ::= & \mathsf{H}(G, e) \mid \mathsf{WS}(x; V) \mid \mathsf{Indis}(x; V_1; V_2) \\ \varphi & ::= & \mathsf{true} \mid \psi \mid \varphi \wedge \varphi \end{array}$$

- **H**(G, e): Not-Hashed-Yet

  $\Pr[S \xleftarrow{r} X : S(e) \in S(\mathcal{T}_H).dom]$ is negligible.

- **WS**$(x; V)$: cannot to compute some "hidden" value.

  $\Pr[S \xleftarrow{r} X : A(S) = S(x)]$ is negligible.

- **Indis**$(x; V_1; V_2)$: seeing V_1 and $f(V_2)$.

But more than 30 rules

Verification Technique: Hoare Logic

Set of rules $(R_i) : \{P\}$ cmd $\{Q\}$

Verification Technique: Hoare Logic

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$c_1$$
$$c_2$$
$$\vdots$$
$$c_n$$

Verification Technique: Hoare Logic

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$\begin{array}{ll} \{P_0\} & c_1 \\ & c_2 \\ & \vdots \\ & c_n \end{array}$$

Verification Technique: Hoare Logic

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$\begin{array}{ll} \{P_0\} & c_1 \\ & c_2 \\ & \vdots \\ & c_n \ \{Indis(out_e)\}\ ? \end{array}$$

Verification Technique: Hoare Logic

Set of rules $(R_i) : \{P\}$ cmd $\{Q\}$

$$
\begin{array}{ll}
(R_5)\{P_0\} & c_1\ \{Q_0\} \\
 & c_2 \\
 & \vdots \\
 & c_n\ \{Indis(out_e)\}\ ?
\end{array}
$$

Verification Technique: Hoare Logic

Set of rules $(R_i) : \{P\}\ \text{cmd}\ \{Q\}$

$$
\begin{array}{l}
(R_5)\{P_0\}\ c_1\ \{Q_0\} \\
(R_2)\{P_1\}\ c_2\ \{Q_2\}, \text{ where } P_1 \subseteq Q_0 \\
\qquad\qquad \vdots \\
\qquad\qquad c_n\ \{Indis(out_e)\}\ ?
\end{array}
$$

Verification Technique: Hoare Logic

Set of rules $(R_i) : \{P\}$ cmd $\{Q\}$

$(R_5)\{P_0\}\ c_1\ \{Q_0\}$
$(R_2)\{P_1\}\ c_2\ \{Q_2\}$, where $P_1 \subseteq Q_0$
$\vdots$
$(R_8)\{P_n\}\ c_n\ \{Indis(out_e)\}$?

Verification Technique: Hoare Logic

Set of rules $(R_i) : \{P\}$ cmd $\{Q\}$

$(R_5)\{P_0\}\ c_1\ \{Q_0\}$
$(R_2)\{P_1\}\ c_2\ \{Q_2\}$, where $P_1 \subseteq Q_0$
$\vdots$
$(R_8)\{P_n\}\ c_n\ \{Indis(out_e)\}$?

Examples of rules:

(X2): $\{\mathrm{Indis}(w; V_1, y, z; V_2)\}\ \ x := y \oplus z\ \ \{\mathrm{Indis}(w; V_1, x, y, z; V_2)\}$

(H6): $\{\mathrm{WS}(y; V_1; V_2, y) \wedge \mathrm{H}(H, y)\}\ \ x := H(y)\ \ \{\mathrm{WS}(y; V_1, x; V_2, y)\}$

Example : Bellare & Rogaway's 1993

$$
\begin{array}{lcr}
r \xleftarrow{r} \{0,1\}^{n_0} & - & \mathrm{Indis}(r) \wedge \mathrm{H}(G, r) \wedge \mathrm{H}(H, h||r) \\
a := f(r) & - & \mathrm{Indis}(a; \mathrm{Var} - r) \wedge \mathrm{WS}(r; \mathrm{Var} - r) \wedge \\
 & - & \mathrm{H}(H, h||r) \\
g := G(r) & - & \mathrm{Indis}(a; \mathrm{Var} - r) \wedge \mathrm{Indis}(g; \mathrm{Var} - r) \wedge \\
 & - & \mathrm{WS}(r; \mathrm{Var} - r) \wedge \mathrm{H}(H, h||r) \\
e := h \oplus g & - & \mathrm{Indis}(a; \mathrm{Var} - r) \wedge \mathrm{Indis}(e) \wedge \\
 & - & \wedge \mathrm{WS}(r; \mathrm{Var} - r) \wedge \mathrm{H}(H, h||r) \\
d := h||r & - & \mathrm{Indis}(a) \wedge \mathrm{Indis}(e) \wedge \\
 & - & \mathrm{WS}(r; \mathrm{Var} - r) \wedge \\
 & - & \mathrm{H}(H, d) \wedge \mathrm{WS}(d) \\
c := H(d) & - & \mathrm{Indis}(a) \wedge \mathrm{Indis}(e) \\
 & - & \wedge \mathrm{Indis}(c) \\
\mathrm{out}_e := a||e||c & - & \mathrm{Indis}(oute; \{\mathrm{in}_e, \mathrm{out}_e\})
\end{array}
$$

Conclusion: Hoare Logics for proving

- **Asymmetric Encryption Schemes**
  - An OCAML prototype of our 30 rules
  - Extensions done for proving IND-CCA using IND-CPA + Plaintext Awareness
  - Exact Security
- **Symmetric Encryption Modes**
  - Counters
  - FOR loops
  - Exact Security
  - An OCAML prototype of our 21 rules
- **Message Authentication Codes (MACs)**
  - Different property: *Unforgeability*
  - Almost-universal Hash function
  - Keep track of possible collisions
  - FOR loops
  - An OCAML prototype of our 44 rules

Outline

Revisited [Benaloh'94] Homomorphic Encryption

$\{0\}_{pk_S}$  $\{1\}_{pk_S}$

Revisited [Benaloh'94] Homomorphic Encryption

$\{0\}_{pk_S}$  $\{1\}_{pk_S}$

$$\prod_{i=1}^{n}\{v_i\}_{pk_S} = \{\sum_{i=1}^{n} v_i\}_{pk_S}$$

Revisited [Benaloh'94] Homomorphic Encryption

$\{0\}_{pk_S}$  $\{1\}_{pk_S}$

$$\prod_{i=1}^{n}\{v_i\}_{pk_S} = \{\sum_{i=1}^{n} v_i\}_{pk_S}$$

Result [FLA'11]

- Original Benaloh's scheme is **ambiguous (33%)**:

  $$dec(enc(14, pk_S), sk_S) = 14 \mod 15 \text{ or } 14 \mod 5 = 4$$

- Proposition of **corrected version**
- Proof using Kristian Gjosteen result

Revisited [Benaloh'94] Homomorphic Encryption

$\{0\}_{pk_S}$  $\{1\}_{pk_S}$

$$\prod_{i=1}^{n}\{v_i\}_{pk_S} = \{\sum_{i=1}^{n} v_i\}_{pk_S}$$

Result [FLA'11]

- Original Benaloh's scheme is **ambiguous (33%)**:

  $dec(enc(14, pk_S), sk_S) = 14 \mod 15 \text{ or } 14 \mod 5 = 4$

- Proposition of **corrected version**
- Proof using Kristian Gjosteen result

Impact on an election: Result can change (either 14 or 4)

Security Properties of E-Voting Protocols

Fairness

Individual Verifiability

Eligibility

Universal Verifiability

Correctness

Privacy

Receipt-Freeness

Robustness

Coercion-Resistance

Security Properties of E-Voting Protocols

Fairness

Individual Verifiability

Eligibility

Universal Verifiability

Correctness

Privacy

Receipt-Freeness

Robustness

Coercion-Resistance

Motivation

Existing several models for Privacy, but they

- designed for a specific type of protocol
- often cannot be applied to other protocols

Motivation

Existing several models for Privacy, but they

- designed for a specific type of protocol
- often cannot be applied to other protocols

Our Contributions:

- Define **fine-grained** Privacy definitions to **compare** protocols
- Analyze **weighted votes** protocols
- **One coercer is enough**

4 Dimensions for Privacy [DLL'12a, DLL'11]

Modeling in Applied π-Calculus

1. **Communication btwn the attacker & the targeted voter**

   [DKR09]  

   Vote-Privacy (VP) Receipt-Freeness (RF) Coercion-Resistance (CR)

2. **Intruder is controlling another voter**

   Outsider (O)  Insider (I)

3. **Secure against Forced-Abstention**: (FA) or not (PO)

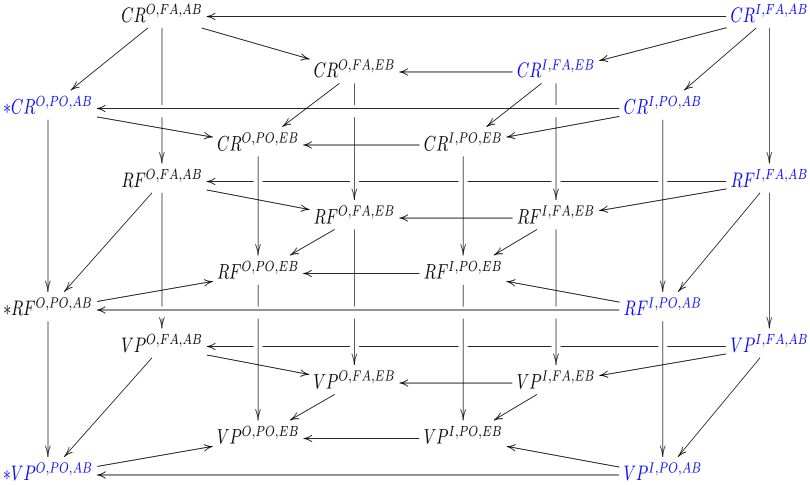
4. **Honest voters behavior**:

   $\exists$  $\forall$

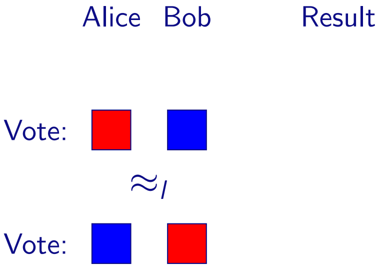
Relations without ∃ and ∀

$$
\begin{tikzcd}
 & CR^{O,FA} \arrow[dl] \arrow[dd] & & CR^{I,FA}\ast \arrow[ll] \arrow[dl] \arrow[dd] \\
\star CR^{O,PO}[\mathrm{LBD^{+}04}] \arrow[dd] & & CR^{I,PO}[\mathrm{BMQR07}] \arrow[ll] \arrow[dd] & \\
 & RF^{O,FA} \arrow[dl] \arrow[dd] & & RF^{I,FA}\diamond \arrow[ll] \arrow[dl] \arrow[dd] \\
\star RF^{O,PO} \arrow[dd] & & RF^{I,PO}[\mathrm{Oka96}] \arrow[ll] \arrow[dd] & \\
 & VP^{O,FA} \arrow[dl] & & VP^{I,FA}\dagger \arrow[ll] \arrow[dl] \\
\star VP^{O,PO}\bullet & & VP^{I,PO}[\mathrm{FOO92}] \arrow[ll] &
\end{tikzcd}
$$

All relations among the notions



Privacy for Weighted Votes [DLL'12b]



Privacy for Weighted Votes [DLL'12b]

	Alice	Bob	Result
	66%	34%	
Vote:			
	$\approx_l$		
Vote:			

Privacy for Weighted Votes [DLL'12b]

	Alice	Bob	Result
	66%	34%	
Vote:			66%, 34%
	$\approx_l$		
Vote:			34%, 66%

Privacy for Weighted Votes [DLL'12b]

	Alice	Bob	Result
	66%	34%	
Vote:			66%, 34%
	$\approx_l$		$\neq$
Vote:			34%, 66%

Privacy for Weighted Votes [DLL'12b]

	Alice	Bob	Result
	66%	34%	
Vote:			66%, 34%
	$\not\approx_l$		$\neq$
Vote:			34%, 66%

 Privacy for Weighted Votes [DLL'12b]

Still: Some privacy is possible!

	Alice	Bob	Carol	Result
	50%	25%	25%	
Vote:				
Vote:				

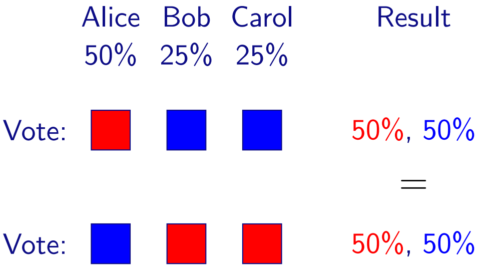
Privacy for Weighted Votes [DLL'12b]

Still: Some privacy is possible!

	Alice 50%	Bob 25%	Carol 25%	Result
Vote:				50%, 50%
Vote:				50%, 50%

Privacy for Weighted Votes [DLL'12b]

Still: Some privacy is possible!



Privacy for Weighted Votes [DLL'12b]

Still: Some privacy is possible!

	Alice	Bob	Carol	Result
	50%	25%	25%	
Vote:				50%, 50%
		$\approx_l$		$=$
Vote:				50%, 50%

Definition of Vote-Privacy (VP) for weighted votes

Idea: Two instances with the same result should be bi-similar

$$
\begin{array}{rccccc}
 & \text{Alice} & \text{Bob} & \cdots & & \text{Result} \\
\text{Vote 1:} & \boxed{V_A^1} & \boxed{V_B^1} & \cdots & & \boxed{\text{Result 1}} \\
 & \approx_l & & & \Leftarrow & \stackrel{?}{=} \\
\text{Vote 2:} & \boxed{V_A^2} & \boxed{V_A^2} & \cdots & & \boxed{\text{Result 2}}
\end{array}
$$

Single-Voter Receipt Freeness (SRF)

Mallory

Alice Bob $\cdots$ Result

$$\begin{array}{cccccc} \boxed{V_A^1} & \boxed{V_B^1} & \cdots & & \boxed{\text{Result 1}} \\ \multicolumn{2}{c}{\approx_l} & & \Leftarrow & \stackrel{?}{=} \\ \boxed{V_A^2} & \boxed{V_B^2} & \cdots & & \boxed{\text{Result 2}} \end{array}$$

Single-Voter Receipt Freeness (SRF)

Mallory		Alice	Bob	$\cdots$		Result
$\boxed{\text{Secret Data}}$	$\leftarrow$	$\boxed{V_A^1}$	$\boxed{V_B^1}$	$\cdots$		$\boxed{\text{Result 1}}$
		$\approx_l$			$\Leftarrow$	$\stackrel{?}{=}$
$\boxed{\text{Fake Data}}$	$\leftarrow$	$\boxed{V_A^2}$	$\boxed{V_B^2}$	$\cdots$		$\boxed{\text{Result 2}}$

Single-Voter Receipt Freeness (SRF)

$$
\begin{array}{ccccccc}
\text{Mallory} & & \text{Alice} & \text{Bob} & \cdots & & \text{Result} \\
\boxed{\text{Secret Data}} & \longleftarrow & \boxed{V_A^1} & \boxed{V_B^1} & \cdots & & \boxed{\text{Result 1}} \\
 & & \multicolumn{2}{c}{\approx_l} & & \Leftarrow & \stackrel{?}{=} \\
\boxed{\text{Fake Data}} & \longleftarrow & \boxed{V_A^2} & \boxed{V_B^2} & \cdots & & \boxed{\text{Result 2}}
\end{array}
$$

If a protocol respects (EQ), then (SRF) and (SwRF) are equivalent.

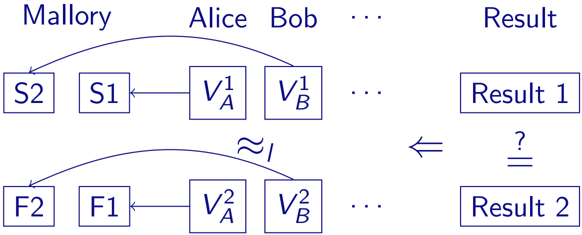
Multi-Voter Receipt Freeness (MRF)

Mallory	Alice	Bob	$\cdots$		Result
$\boxed{S1} \leftarrow$	$\boxed{V_A^1}$	$\boxed{V_B^1}$	$\cdots$		$\boxed{\text{Result 1}}$
	$\approx_l$			$\Leftarrow$	$\stackrel{?}{=}$
$\boxed{F1} \leftarrow$	$\boxed{V_A^2}$	$\boxed{V_B^2}$	$\cdots$		$\boxed{\text{Result 2}}$

Multi-Voter Receipt Freeness (MRF)

$$\begin{tikzcd}
\text{Mallory} & & \text{Alice} & \text{Bob} & \cdots & \text{Result} \\
\boxed{S2} & \boxed{S1} & \boxed{V_A^1} \arrow[l] & \boxed{V_B^1} \arrow[lll, bend right] & \cdots & \boxed{\text{Result 1}} \\
 & & & \approx_l & \Leftarrow & \stackrel{?}{=} \\
\boxed{F2} & \boxed{F1} & \boxed{V_A^2} \arrow[l] & \boxed{V_B^2} \arrow[lll, bend right] & \cdots & \boxed{\text{Result 2}}
\end{tikzcd}$$

Multi-Voter Receipt Freeness (MRF)



(MRF) implies (SRF) and (MCR) implies (SCR).

One Coerced Voter is enough!

$$
\begin{tikzcd}
\mathrm{SwCR} \arrow[r, leftrightarrow, "\mathrm{EQ}"] \arrow[d] & \mathrm{SCR} \arrow[d] \arrow[r, bend left, "\mathrm{Cor,\ Mod}"] & \mathrm{MCR} \arrow[l] \arrow[d] \\
\mathrm{SwRF} \arrow[r, leftrightarrow, "\mathrm{EQ}"] \arrow[d] & \mathrm{SRF} \arrow[d] \arrow[r, bend left, "\mathrm{Cor,\ Mod}"] & \mathrm{MRF} \arrow[l] \\
\mathrm{SwP} \arrow[r, leftrightarrow, "\mathrm{EQ}"] & \mathrm{VP} &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathrm{EQ} \arrow[d] \\
\mathrm{Cor}
\end{tikzcd}
$$

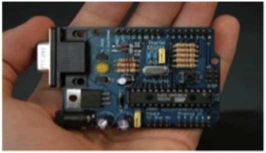
Unique decomposition of processes in the applied π-calculus.

Outline

Challenges in WSNs

Nodes

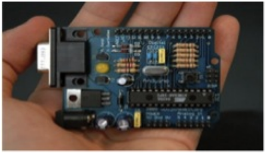
- Broadcast communication
- Low computation power
- Battery



Challenges in WSNs

Nodes

- Broadcast communication
- Low computation power
- Battery



- **Cryptography:** Lightweight, energy- and resource-aware …
- **Properties:** (k)-neighborhood, routing …
- **Intruders:** Black-hole, wormhole, Byzantine, independent …

Our Contributions

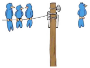
- (k)-Neighbourhood Verification [JL'12]
- Independent Intruders [KL'12]
- Analysis of non-backtracking random walk [ADGL'12]
- Resilient routing algorithm [ADJL]

Our Contributions

- (k)-Neighbourhood Verification [JL'12]
- Independent Intruders [KL'12]
- Analysis of non-backtracking random walk [ADGL'12]
- Resilient routing algorithm [ADJL]

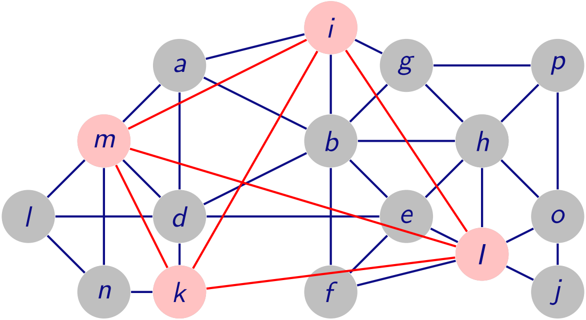
Usual Intruders

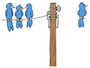
Dolev-Yao's Intruder [83]



Intruder Model in WSNs

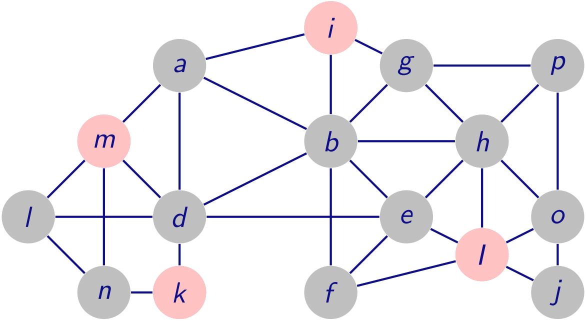
Several intruders with sharing [ACD12]





Independent Intruder Model

Independent intruders without sharing



Usual Constraints System

$$
\begin{array}{ccc}
T_1 & \Vdash & u_1 \\
T_2 & \Vdash & u_2 \\
 & \vdots & \\
T_n & \Vdash & u_n
\end{array}
$$

- **Intruder knowledge monotonicity**: $T_1 \subseteq \cdots \subseteq T_n$.
- **Variable origination**: if x occurs in $vars(T_i)$ for certain T_i then there exists $k < i$ such that $x \in vars(u_k)$.

Partially Well-Formed Constraint System

Partially well-formed constraints system

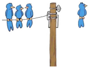
$$\mathcal{C} = T_1^l \Vdash u_1 \wedge \cdots \wedge T_n^q \Vdash u_n$$

- Global Origination.
- Partial monotonicity:
  $T_k^j \subseteq T_i^j$ for every $j \in \{1, 2, \ldots, m\}$ such that $k < i$.

Quasi-Solved Form

$$
\begin{array}{lll}
R_{ax} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \mathcal{C} & \text{if } T_i^j \cup \{x \mid T_k^j \Vdash x \in \mathcal{C}, k < i\} \vdash u_i \\
R_{unif} : & \mathcal{C} \rightsquigarrow_\sigma \mathcal{C}\sigma & \sigma = mgu(t_1, t_2),\ t_1, t_2 \in st(\mathcal{C}) \\
R'_{unif} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T_i^j\sigma \Vdash u_i\sigma & \sigma = mgu(t, f(t_1, t_2)), f \in \{\langle -, - \rangle, - :: -\}, \\
 & & t \in vars(u_i), t_1, t_2 \in st(T_k^l),\ \text{where } k \leq i \\
R_f : & \mathcal{C} \wedge T^j \Vdash f(u, v) \rightsquigarrow \mathcal{C} \wedge T^j \Vdash u \wedge T^j \Vdash v & \text{if } f \in \{senc, aenc, \langle -, - \rangle, - :: -, hmac, sig\} \\
R_{fail} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \perp & \text{if } T_i^j = \emptyset, \text{or } vars(T_i^j \cup \{u_i\}) = \emptyset \\
 & & \text{and } T_i^j \nvdash u_i
\end{array}
$$

Soundness, completeness and termination.



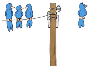
Quasi-Solved Form

$$
\begin{array}{lll}
R_{ax} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \mathcal{C} & \text{if } T_i^j \cup \{x \mid T_k^j \Vdash x \in \mathcal{C}, k < i\} \vdash u_i \\
R_{unif} : & \mathcal{C} \rightsquigarrow_\sigma \mathcal{C}\sigma & \sigma = mgu(t_1, t_2),\ t_1, t_2 \in st(\mathcal{C}) \\
R'_{unif} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T_i^j\sigma \Vdash u_i\sigma & \sigma = mgu(t, f(t_1, t_2)), f \in \{\langle -, - \rangle, - :: -\}, \\
 & & t \in vars(u_i), t_1, t_2 \in st(T_k^l),\ \text{where } k \leq i \\
R_f & : \mathcal{C} \wedge T^j \Vdash f(u, v) \rightsquigarrow \mathcal{C} \wedge T^j \Vdash u \wedge T^j \Vdash v & \text{if } f \in \{senc, aenc, \langle -, - \rangle, - :: -, hmac, sig\} \\
R_{fail} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \perp & \text{if } T_i^j = \emptyset, \text{or } vars(T_i^j \cup \{u_i\}) = \emptyset \\
 & & \text{and } T_i^j \nvdash u_i
\end{array}
$$

Soundness, completeness and termination.

Example of Quasi-Solved Form:

$$
\begin{aligned}
T_1^1 &= \{a, b\} \Vdash x \\
T_2^2 &= \{x\} \Vdash a \\
T_3^3 &= \{x\} \Vdash b
\end{aligned}
$$



Quasi-Solved Form

$$
\begin{array}{lll}
R_{ax} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \mathcal{C} & \text{if } T_i^j \cup \{x \mid T_k^j \Vdash x \in \mathcal{C}, k < i\} \vdash u_i \\
R_{unif} : & \mathcal{C} \rightsquigarrow_\sigma \mathcal{C}\sigma & \sigma = mgu(t_1, t_2),\ t_1, t_2 \in st(\mathcal{C}) \\
R'_{unif} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow_\sigma \mathcal{C}\sigma \wedge T_i^j\sigma \Vdash u_i\sigma & \sigma = mgu(t, f(t_1, t_2)), f \in \{\langle -, - \rangle, - :: -\}, \\
 & & t \in vars(u_i), t_1, t_2 \in st(T_k^l),\ \text{where } k \leq i \\
R_f & : \mathcal{C} \wedge T^j \Vdash f(u, v) \rightsquigarrow \mathcal{C} \wedge T^j \Vdash u \wedge T^j \Vdash v & \text{if } f \in \{senc, aenc, \langle -, - \rangle, - :: -, hmac, sig\} \\
R_{fail} : & \mathcal{C} \wedge T_i^j \Vdash u_i \rightsquigarrow \perp & \text{if } T_i^j = \emptyset, \text{or } vars(T_i^j \cup \{u_i\}) = \emptyset \\
 & & \text{and } T_i^j \nvdash u_i
\end{array}
$$

Soundness, completeness and termination.

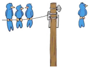
Example of Quasi-Solved Form:

$$
\begin{array}{rcl}
T_1^1 & = & \{a, b\} \Vdash x \\
T_2^2 & = & \{x\} \Vdash a \\
T_3^3 & = & \{x\} \Vdash b
\end{array}
$$

Procedure for finding a solution to a quasi-solved form.

Resilient Routing Algorithms

Even with "**perturbation**" a **resilient** protocol should work "**well**"



Resilient Routing Algorithms

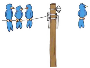
Even with "**perturbation**" a **resilient** protocol should work "**well**"

- Perturbation: abnormal behavior, node destruction, battery ...

Resilient Routing Algorithms

Even with "**perturbation**" a **resilient** protocol should work "**well**"

- Perturbation: abnormal behavior, node destruction, battery ...
- Well: Hitting time, average delivery rate...



Resilient Routing Algorithms

Even with "**perturbation**" a **resilient** protocol should work "**well**"

- Perturbation: abnormal behavior, node destruction, battery ...
- Well: Hitting time, average delivery rate...

Existing protocols

Probabilistic *vs* **Deterministic**

Random walk GBR, GFG

Resilient Routing Algorithms

Even with "**perturbation**" a **resilient** protocol should work "**well**"

- Perturbation: abnormal behavior, node destruction, battery ...
- Well: Hitting time, average delivery rate...

Existing protocols

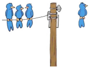
Probabilistic *vs* **Deterministic**
Random walk GBR, GFG

Our Goal: Design an efficient resilient routing algorithm using a reputation mechanism

Our Resilient Algorithm: TLCNS [ADJL]

Shared symmetric key K_{OS} between the sink and all nodes O.

- Each node O sends: $\{Data, N_O\}_{K_{OS}}, H(N_O), O, F$
- Sink S acknowledges: N_O, O



Our Resilient Algorithm: TLCNS [ADJL]

Shared symmetric key K_{OS} between the sink and all nodes O.

- Each node O sends: $\{Data, N_O\}_{K_{OS}}, H(N_O), O, F$
- Sink S acknowledges: N_O, O

3 lists for each node:

- $M_{ack} = [(H(N_O), A), (H(N_B), C)]$: List of hashed nonces and sender identity.
- $M_{Queue} = [(N_O^1, A), (N_O^2, B)]$: List of messages sent
- $L_{Routing} = [A, B, C]$: List of "preferred" first hops (FIFO)

Our Resilient Algorithm: TLCNS [ADJL]

Shared symmetric key K_{OS} between the sink and all nodes O.

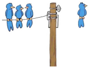
- Each node O sends: $\{Data, N_O\}_{K_{OS}}, H(N_O), O, F$
- Sink S acknowledges: N_O, O

3 lists for each node:

- $M_{ack} = [(H(N_O), A), (H(N_B), C)]$: List of hashed nonces and sender identity.
- $M_{Queue} = [(N_O^1, A), (N_O^2, B)]$: List of messages sent
- $L_{Routing} = [A, B, C]$: List of "preferred" first hops (FIFO)

Why does it work?

- Each node prefers preferred next hop
- All neighbours are possible



Scenario for testing the Resilience

- Simulation using SINALGO
- $|L_{Routing}| = 10$, $|M_{Queue}| = 5$ and $|M_{ack}| = 3$
- 200 nodes, 1 sink

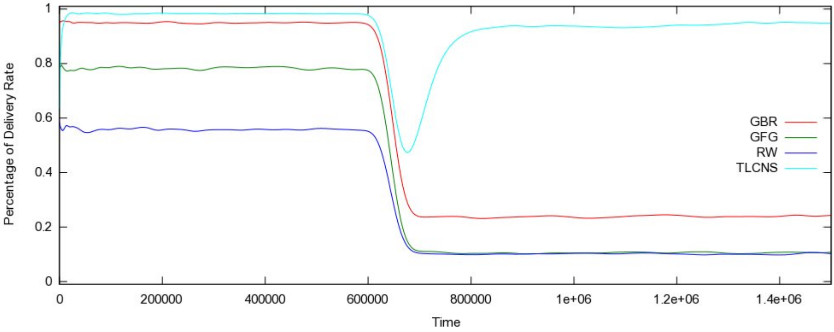
Intruders:

- Black Holes: Node not forwarding any message
- Worm Holes: False link in the topology

Scenario in 2 phases:

- Static: 10 Black holes + 10 Wormholes
- Dynamic: 20 Black holes
  (Wormholes $\rightarrow$ Black Holes)

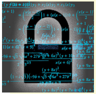
Results


Percentage of Delivery Rate
Time
0
0.2
0.4
0.6
0.8
1
0
200000
400000
600000
800000
1e+06
1.2e+06
1.4e+06
GBR
GFG
RW
TLCNS

Outline

Summary

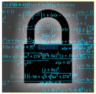
Automatic proofs of programs (Hoare Logic)



- Generic Asymmetric Encryption [CDELL'08, CDELL'10]
- Generic Encryption Mode: counter + For loop [GLL'09]
- Generic MAC: Double execution + For loop [GLL'13]

Summary

Automatic proofs of programs (Hoare Logic)



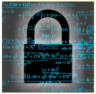
- Generic Asymmetric Encryption [CDELL'08, CDELL'10]
- Generic Encryption Mode: counter + For loop [GLL'09]
- Generic MAC: Double execution + For loop [GLL'13]

Cryptography & Process Algebra (Applied π-Calculus)

- Revisited Benaloh's encryption scheme [FLA'11]
- Privacy notions [DLL'12a, DLL'11]
- Weighted votes [DLL'12b]

Summary

Automatic proofs of programs (Hoare Logic)

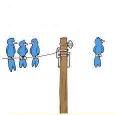


- Generic Asymmetric Encryption [CDELL'08, CDELL'10]
- Generic Encryption Mode: counter + For loop [GLL'09]
- Generic MAC: Double execution + For loop [GLL'13]

Cryptography & Process Algebra (Applied π-Calculus)

- Revisited Benaloh's encryption scheme [FLA'11]
- Privacy notions [DLL'12a, DLL'11]
- Weighted votes [DLL'12b]

Constraints Solving & Randomized Algorithms



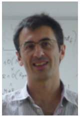
- Neighbourhood Discovery Verification [JL'12]
- Independent Intruders [KL'12]
- Design of routing algorithms [AGDL'12, ADLP'11]

Future Work

- **Computer-Aided Cryptography**:
  - Hoare Logic for other primitives: Pairing, E-Stream ...
  - How to prove Benaloh' scheme?
  - Using verification for the synthesis of new schemes
- **Properties**:
  - E-auctions: Non cancellation, Non repudiation, Privacy ...
  - Non-functional properties for WSNs: energy consumption.
- **Intruder Model**:
  - With a battery
  - Mobility

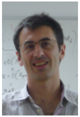
Thanks



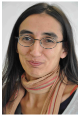
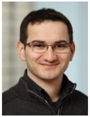




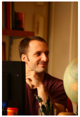
Thanks

Questions ?



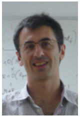




Thanks


MD RJ NA. . .








Questions ?


Thanks for your attention

Questions ?

Loop Analysis

Only one for rule

(F1) $\{\psi(p-1)\}$ for $k = p$ to q do: $[c_k]$ $\{\psi(q)\}$
provided $\{\psi(k-1)\}$ c_k $\{\psi(k)\}$ for $p \leq k \leq q$

Loop Analysis

Only one for rule

(F1) $\{\psi(p-1)\}$ for $k = p$ to q do: $[c_k]$ $\{\psi(q)\}$
provided $\{\psi(k-1)\}$ c_k $\{\psi(k)\}$ for $p \leq k \leq q$

How do you find such loop invariant ?

Loop Analysis

Only one **for** rule

(F1) $\{\psi(p-1)\}$ for $k=p$ to q do: $[c_k]$ $\{\psi(q)\}$
provided $\{\psi(k-1)\}$ c_k $\{\psi(k)\}$ for $p \leq k \leq q$

How do you find such loop invariant ?

First approach

- A backward analysis gives: $\{\psi(k-1)\}$ c_k $\{\varphi(k)\}$
- If $\varphi(k) \wedge \psi(k) \Rightarrow \varphi(k)$ we have our invariant
- Otherwise, try with $\varphi'(k) = \varphi(k) \wedge \psi(k)$ ...

Loop Analysis

$$\{\psi_1(l-1)\}\ c_l\ \{\varphi(l)\}$$

Try one step further

$$\{\psi_2(l-1)\}\ c_l\ \{\varphi(l) \wedge \psi_1(l)\}$$

Loop Analysis

$$\{\psi_1(l-1)\}\ c_l\ \{\varphi(l)\}$$

Try one step further

$$\{\psi_2(l-1)\}\ c_l\ \{\varphi(l) \wedge \psi_1(l)\}$$

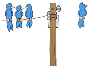
Analyzing : $\varphi(l)$, $\varphi(l) \wedge \psi_1(l)$ and $\varphi(l) \wedge \psi_1(l) \wedge \psi_2(l)$.
we identify $\gamma(l)$ such that:

- $\gamma(l)$ appears in $\varphi(l)$,
- $\gamma(l-1)$ appears in $\psi_1(l)$
- $\gamma(l-2)$ appears in $\psi_2(l)$.

We then use $\varphi'(l) = \varphi(l) \wedge \bigwedge_{j=p-1}^{j=l-1} \gamma(j)$

Tabu List in the Message [SIROCCO'12]

GOAL : Avoiding Cycles!



Tabu List in the Message [SIROCCO'12]

GOAL : Avoiding Cycles!

HOW :

- Store IDs of visited nodes in message.
- Which Update (Rand, FIFO, LRU)? Size?

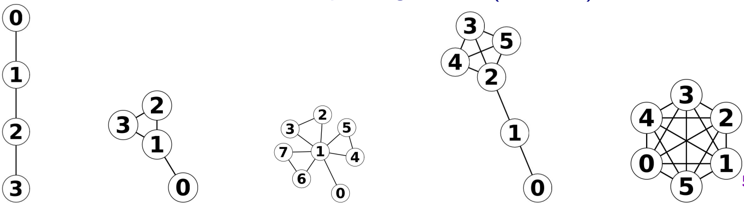
Tabu List in the Message [SIROCCO'12]

GOAL : Avoiding Cycles!

HOW :

- Store IDs of visited nodes in message.
- Which Update (Rand, FIFO, LRU)? Size?

Surprising: TLM(1,FIFO) is the best



Message Authentication Code

What is a MAC?

- MAC algorithm takes a key and a message ($\{0,1\}^*$) and outputs a *tag*
- Purpose: ensure integrity and authenticity of messages
- Upon receiving (m, tag), receiver computes $tag' = MAC(k, m)$ and accepts the message as authentic if $tag = tag'$

Message Authentication Code

What is a MAC?

- MAC algorithm takes a key and a message ($\{0,1\}^*$) and outputs a *tag*
- Purpose: ensure integrity and authenticity of messages
- Upon receiving (m, tag), receiver computes $tag' = MAC(k, m)$ and accepts the message as authentic if $tag = tag'$

Applications

- authenticity
- IND-CCA security (encrypt-then-mac construction)
- building block for many other cryptographic protocols

MAC Security

For a message authentication code MAC define the experiment Exp_{MAC} as follows:

- Sample $k \xleftarrow{R} \{0,1\}^{\eta}$.
- $(m^*, tag) \xleftarrow{R} \mathcal{A}^{MAC(k,\cdot)}(\eta)$
- if $MAC(k, m^*) = tag$ and $\mathcal{A}$ never queried m^* to its $MAC(k, \cdot)$ oracle, return 1, else return 0

Definition

$\textsc{Adv}_{\mathcal{A}}^{UNF}(\eta) = Pr[Exp_{MAC} = 1]$

A MAC is existentially unforgeable if $\textsc{Adv}_{\mathcal{A}}^{UNF}(\eta)$ is negligible for every polynomial-time adversary $\mathcal{A}$.

Challenges with MACs

- Security property cannot be modeled as simply as for encryption
- MACs are deterministic
- Messages with common prefixes will cause same queries to the block cipher

Challenges with MACs

- Security property cannot be modeled as simply as for encryption
- MACs are deterministic
- Messages with common prefixes will cause same queries to the block cipher

$\Rightarrow$ We need fundamentally new trick

Security Proofs of MACs

Usual method for proving MAC security

- Pseudo-random functions are good MACs

$\Rightarrow$ prove that the compressing part of the MAC is an almost-universal hash function

$\Rightarrow$ combine that almost-universal hash with a mixing step to get a PRF

Definition

Almost-universal hash function A hash function family $\{H_k\}_{k\in\{0,1\}^\eta}$ is an *almost-universal hash function* family if for any two messages $m_0, m_1 \in \{0,1\}^*$, $\Pr[H_k(m_0) = H_k(m_1)]$ is negligible, where the probability is taken over the choice of the key.

Predicates

Equal(x, y): the probability that $S(x) \neq S'(y)$ is negligible.

Unequal(x, y): the probability that $S(x) = S'(y)$ is negligible.

E$(\mathcal{E}, x; V)$: the probability that the value of x is either in $\mathcal{L}_{\mathcal{E}}$ or in V is negligible.

H$(\mathcal{H}, x; V)$: the probability that the value of x is either in $\mathcal{L}_{\mathcal{H}}$ or in V is negligible.

Empty: that the probability that $\mathcal{L}_{\mathcal{E}}$ contains an element is negligible.

Indis$(x; V; V')$: the value of x is indistinguishable from a random value given the values of the variables in V in this execution and the values of the variables in V' from the parallel execution,

Our Strategy

Two Step Strategy

- Hoare logic to prove 'front-end' is almost-universal
  - take advantage of non-adaptive adversary
  - empty list of block cipher queries at the beginning
  - consider two simultaneous executions of the code
  - examine probability of collisions between intermediate values
- List of possible 'mixing steps'

$$\begin{aligned} \mathrm{c} \quad ::= \quad & x := \rho^i(y) \mid x := \mathcal{E}(y) \mid x := \mathcal{H}(y) \mid x := y \oplus z \mid x := y \| z \mid \\ & x := y \mid \text{for} \ \ x = i \ \ \text{to} \ \ j \ \ \text{do:} \ \ c_x \mid c_1 ; c_2 \end{aligned}$$