Special topics in Logic and Security I

Master Year II, Sem. I, 2022-2023

Ioana Leuștean
FMI, UB

The Dolev-Yao attacker

Recall that we defined the attacker knowledge using a deductive systems, e.g.

if	$M \vdash t$ and $M \vdash k$	then	$M \vdash \{\!\|\, t \,\|\!\}_k$
if	$M \vdash t_1$ and $M \vdash t_2$	then	$M \vdash (t_1, t_2)$
if	$M \vdash \{\!\|\, t \,\|\!\}_k$ and $M \vdash k$	then	$M \vdash t$
if	$M \vdash (t_1, t_2)$	then	$M \vdash t1$ and $M \vdash t_2$
...			

where $M \vdash t$ means that t can be deduced knowing M

The Dolev-Yao attacker

Recall that a *Horn clause* has the form

$$H_1 \wedge \cdots \wedge H_n \to C$$

where $H_1, \ldots, H_n$ and C are atomic formulas in FOL.

We consider a unary predicate `att` such that

$$att(t) \text{ means that the attacker knows } t.$$

The attacker deduction system can be defined using implications as follows:

$$\begin{array}{l}
att(t) \wedge att(k) \to att(\{\!|\, t \,|\!\}_k) \\
att(t_1) \wedge att(t_2) \to att((t_1, t_2)) \\
att(\{\!|\, t \,|\!\}_k) \wedge att(k) \to att(t) \\
att(t_1, t_2) \to att(t_1) \\
att(t_1, t_2) \to att(t_2)
\end{array}$$

which are obviously Horn clauses.

The Dolev-Yao attacker

$$\begin{array}{l} att(t) \wedge att(k) \rightarrow att(\{\!| t |\!\}_k) \\ att(t_1) \wedge att(t_2) \rightarrow att((t_1, t_2)) \\ att(\{\!| t |\!\}_k) \wedge att(k) \rightarrow att(t) \\ att(t_1, t_2) \rightarrow att(t_1) \\ att(t_1, t_2) \rightarrow att(t_2) \end{array}$$

These can be expressed in Prolog as follows:

```
att(senc(X, Y)) :- att(X), att(Y).
att(pair(X,Y)) :- att(X), att(Y).
att(X) :- att(senc(X, Y)), att(Y).
att(X) :- att(pair(X, _))
att(Y) :- att(pair(_, Y)).
```

Note that Prolog (compound) terms - `pair(X,Y)` and `senc(X,Y)` - are used in order to define pairing and symmetric ecryption.

Prolog: the Dolev-Yao attacker

Is Prolog suitable for such inferences?

```
att(ka).
att(kb).
att(senc(secret,pair(ka,kb))).
att(senc(X, Y)) :- att(X), att(Y).
att(pair(X,Y)) :- att(X), att(Y).
att(X) :- att(senc(X, Y)), att(Y).
att(X) :- att(pair(X, _)).
att(Y) :- att(pair(_, Y)).

?- att(secret).
true
```

Prolog: the Dolev-Yao attacker

```
?- trace.
true.

[trace]  ?- att(secret).
   Call: (10) att(secret) ? creep
   Call: (11) att(senc(secret, _21612)) ? creep
   Exit: (11) att(senc(secret, pair(ka, kb))) ? creep
   Call: (11) att(pair(ka, kb)) ? creep
   Call: (12) att(ka) ? creep
   Exit: (12) att(ka) ? creep
   Call: (12) att(kb) ? creep
   Exit: (12) att(kb) ? creep
   Exit: (11) att(pair(ka, kb)) ? creep
   Exit: (10) att(secret) ? creep
true .
```

Prolog: the Dolev-Yao attacker

We can further add rules for public encryption and signing:

```
att(pk(X)) :- att(X).
att(X) :- att(aenc(X, pk(Y))), att(Y).
att(aenc(X, Y)) :- att(X), att(Y).

att(X) :- att(sign(X, Y)).
att(sign(X,Y)) :- att(X), att(Y).

att(X) :- att(senc(X, Y)), att(Y).
att(senc(X, Y)) :- att(X), att(Y).

att(X) :- att(pair(X, _)).
att(Y) :- att(pair(_, Y)).
att(pair(X,Y)) :- att(X), att(Y).
```

Can we follow the same ideas for protocols?

The Denning-Sacco protocol

We consider the Denning-Sacco protocol (without timestamps):

(1) $A \longrightarrow B : \{\!|\ \{\!|\ k\ |\!\}_{sk(A)}\ |\!\}_{pk(B)}$
(2) $B \longrightarrow A : \{\!|\ secret\ |\!\}_{k}$

The protocol is vulnerable to the following attack

(att1) $A \longrightarrow E : \{\!|\ \{\!|\ k\ |\!\}_{sk(A)}\ |\!\}_{pk(E)}$
(att2) $E \longrightarrow B : \{\!|\ \{\!|\ k\ |\!\}_{sk(A)}\ |\!\}_{pk(B)}$
(att3) $B \longrightarrow A : \{\!|\ secret\ |\!\}_{k}$

Since E is a corrupted agent, by (att1) the attacker knows the key k so, by (att3), he knows the *secret*.

The Denning-Sacco protocol

We consider the Denning-Sacco protocol (without timestamps):

(1) $A \longrightarrow B : \{\!| \{\!| k |\!\}_{sk(A)} |\!\}_{pk(B)}$
(2) $B \longrightarrow A : \{\!| secret |\!\}_k$

The communication between A and B cand be represented (from the attacker's point of view) by:

```
att(senc(secret, K)) :- att(aenc(sign(K, skA), pk(skB))).
```

If B receives what (s)he expects, then (s)he behaves following the protocols and the messages are "added" to the adversary knowledge.
Note that the agents are identified with their secret keys.

The Denning-Sacco protocol

(1) $A \longrightarrow B : \{\!|\, \{\!|\, k \,|\!\}_{sk(A)} \,|\!\}_{pk(B)}$
(2) $B \longrightarrow A : \{\!|\, secret \,|\!\}_{k}$

We've defined the fact that B acts when s(he) receives a message, but we still need to define that fact that A starts the protocol. To do this we can think that "A wants to speak to any principal (who's public key is known)":

```
att(aenc(sign(k, skA), pk(X))) :- att(pk(X)).
```

Moreover, we can express the fact that k is a key that depends on the communication partner:

```
att(aenc(sign(k(pk(X)), skA), pk(X))) :- att(pk(X)).
```

Prolog: the Denning-Sacco protocol - naive approach

We've shown that the protocol

(1) $A \longrightarrow B : \{\!|\, \{\!|\, k \,|\!\}_{sk(A)} \,|\!\}_{pk(B)}$

(2) $B \longrightarrow A : \{\!|\, secret \,|\!\}_k$

can be defined in Prolog by:

```
att(aenc(sign(K, skA), pk(X))) :- att(pk(X)).
att(senc(secret, K)) :- att(aenc(sign(K, skA), pk(skB))).
```

Can we find the attack? This means that:

```
?- att(secret).
true
```

Prolog: the Denning-Sacco protocol - naive approach

In order to analyze the protocol we need to represent the attacker inference system, as well as the adversary initial knowledge:

```
% initial knowledge
att(skI).
att(pk(skA)).
att(pk(skB)).

% the attacker inference system
att(pk(X)) :- att(X).
att(X) :- att(aenc(X, pk(Y))), att(Y).
att(aenc(X, Y)) :- att(X), att(Y).
att(X) :- att(sign(X, Y)).
att(sign(X,Y)) :- att(X), att(Y).
att(X) :- att(senc(X, Y)), att(Y).
att(senc(X, Y)) :- att(X), att(Y).

%the protocol
att(aenc(sign(K, skA), pk(X))) :- att(pk(X)).
att(senc(secret, K)) :- att(aenc(sign(K, skA), pk(skB))).
```

Prolog: the Denning-Sacco protocol - naive approach

```
att(skI).
att(pk(skA)).
att(pk(skB)).
att(aenc(sign(K, skA), pk(X))) :- att(pk(X)).
att(senc(secret, K)) :- att(aenc(sign(K, skA), pk(skB))).
att(pk(X)) :- att(X).
att(X) :- att(senc(X, Y)), att(Y).
att(aenc(X, Y)) :- att(X), att(Y).
att(sign(X,Y)) :- att(X), att(Y).
att(X) :- att(aenc(X, pk(Y))), att(Y).
att(X) :- att(sign(X, Y)).
att(senc(X, Y)) :- att(X), att(Y).


?- att(secret).
true
```

Prolog: the Denning-Sacco protocol - naive approach

```
?- trace.
[trace]  ?- att(secret).
  Call: (10) att(secret) ? creep
   Call: (11) att(senc(secret, _16792)) ? creep
   Call: (12) att(aenc(sign(_16792, skA), pk(skB))) ? creep
   Call: (13) att(pk(skB)) ? creep
   Exit: (13) att(pk(skB)) ? creep
   Exit: (12) att(aenc(sign(_16792, skA), pk(skB))) ? creep
   Exit: (11) att(senc(secret, _16792)) ? creep
   Call: (11) att(_16792) ? creep
   Exit: (11) att(skI) ? creep
   Exit: (10) att(secret) ? creep
true
```

Prolog: the Denning-Sacco protocol - naive approach

Why "naive"?

- The clause ordering affects the results or it can provide false attacks.
- The clause ordering affects termination.

However, the idea of using Horn clauses for modeling and analyzing protocols is valuable, and few tools and theories are known.

"The ProVerif tool takes protocols written in a variant of the applied pi calculus as input together with a security property to verify. The protocol is then automatically translated into a set of first-order Horn clauses and the properties are translated into derivability queries. The verification algorithm is based on a dedicated Horn clause resolution procedure."

[1] Véronique Cortier, Steve Kremer. Formal Models and Techniques for Analyzing Security Protocols: A Tutorial. Foundations and Trends in Programming Languages, Now Publishers, 2014, 1 (3), pp.117.
`https://hal.archives-ouvertes.fr/hal-01090874`

References

1 Véronique Cortier, Steve Kremer. Formal Models and Techniques for Analyzing Security Protocols: A Tutorial. Foundations and Trends in Programming Languages, Now Publishers, 2014, 1 (3), pp.117. `https://hal.archives-ouvertes.fr/hal-01090874`

2 Bruno Blanchet. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. Foundations and Trends® in Privacy and Security , Now publishers inc, 2016, 1 (1-2), pp.1 - 135.`https://hal.inria.fr/hal-01423760/`

3 Bruno Blanchet. Using Horn Clauses for Analyzing Security Protocols. In Véronique Cortier and Steve Kremer, editors, Formal Models and Techniques for Analyzing Security Protocols, volume 5 of Cryptology and Information Security Series, pages 86-111. IOS Press, March 2011.

4 Abadi, M., B. Blanchet, and C. Fournet. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. Journal of the ACM Vol. 65Issue 1February 2018 Article No.: 1pp 1–41. Available at http://arxiv.org/abs/1609. 03003v1.

ProVerif: protocols as Horn clauses

Let $\mathcal{F}$ be a *signature* (a set of function symbols together with their arity), $\mathcal{N}$ a set of *names* (representing nonces, keys, identities) and $\mathcal{X}$ a set of variables.

- The *terms* are defined by:

$Term$	$::=$	x, y, z	variables from $\mathcal{X}$
		$a[t_1, \ldots, t_n]$	names from $\mathcal{N}$
		$f(t_1, \ldots, t_n)$	function application for f from $\mathcal{F}$

- The *predicate* is $att(t)$, which is *true* if the attacker knows the message modelled by t.

- The *Horn clauses* are:
  - facts: $att(t)$
  - rules: $att(t_1) \wedge \cdots \wedge att(t_n) \rightarrow att(t)$

ProVerif: the attacker

$att(x) \wedge att(y) \rightarrow att(senc(x,y))$
$att(senc(x,y)) \wedge att(y) \rightarrow att(x)$

$att(x) \rightarrow att(pk(x))$
$att(x) \wedge att(y) \rightarrow att(aenc(x,y))$
$att(aenc(x,pk(y))) \wedge att(y) \rightarrow att(x)$

$att(x) \wedge att(y) \rightarrow att(sign(x,y))$
$att(sign(x,y)) \rightarrow att(x)$

$att(x) \wedge att(y) \rightarrow att((x,y))$
$att(x,y) \rightarrow att(x)$
$att(x,y) \rightarrow att(y)$

$att(t)$ for any term t in the initial knowledge of the attacker
$att(a[])$ for at least one name a

ProVerif: Horn clauses for the Denning-Sacco protocol

proverif2.04\examples\horn\secr

`http://proverif.inria.fr`

```
(* Initialization *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);
(* The attacker *)
c:x & c:encrypt(m,pk(x)) -> c:m;
c:x & c:sencrypt(m,x) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
c:x & c:y -> c:sencrypt(x,y);
c:sign(x,y) -> c:x;
c:x & c:y -> c:sign(x,y);
(* The protocol *)
(* A *)
c:pk(x) -> c:encrypt(sign(k[pk(x)], sA[]), pk(x));
(* B *)
c:encrypt(sign(k, sA[]), pk(sB[])) -> c:sencrypt(secret[], k).
```

ProVerif: Horn clauses for the Denning-Sacco protocol

proverif2.04\examples\horn\secr
`http://proverif.inria.fr`

```
Clause 15: c:x & c:encrypt(m,pk(x)) -> c:m
Clause 14: c:x -> c:pk(x)
Clause 13: c:x & c:y -> c:encrypt(x,y)
Clause 12: c:sign(x,y) -> c:x
Clause 11: c:x & c:y -> c:sign(x,y)
Clause 10: c:pk(x) -> c:encrypt(sign(k[pk(x)],sA[]),pk(x))
Clause 9: c:encrypt(sign(k_1,sA[]),pk(sB[])) ->
                         c:encrypt(secret[],pk(k_1))
Clause 8: c:c[]
Clause 7: c:pk(sA[])
Clause 6: c:pk(sB[])
Clause 5: c:x & c:encrypt(m,pk(x)) -> c:m
Clause 4: c:x -> c:pk(x)
Clause 3: c:x & c:y -> c:encrypt(x,y)
Clause 2: c:sign(x,y) -> c:x
```

ProVerif: Horn clauses for the Denning-Sacco protocol

proverif2.04\examples\horn\secr
`http://proverif.inria.fr`

```
Derivation:
Abbreviations:
k_1 = k[pk(x)]
clause 15 c:secret[]
    clause 12 c:k_1
        duplicate c:sign(k_1,sA[])
    clause 9 c:encrypt(secret[],pk(k_1))
        clause 13 c:encrypt(sign(k_1,sA[]),pk(sB[]))
            clause 15 c:sign(k_1,sA[])
                duplicate c:x
                clause 10 c:encrypt(sign(k_1,sA[]),pk(x))
                    clause 14 c:pk(x)
                        any c:x
            clause 6 c:pk(sB[])

RESULT goal reachable:  c:secret[]
```

ProVerif

- The protocols are specified in a variant of π- **calculus**.
- The description of the protocol is automatically translated in a set of **Horn clauses**.
- The security property is checked by performing a **particular version of resolution**, developed for this particular setting.

A π-calculus for protocols

Let $\mathcal{F}$ be a *signature*, $\mathcal{N}$ a set of *names* and $\mathcal{X}$ a set of variables such that $\mathcal{N}$ and $\mathcal{X}$ contain special subsets of *channel* names and *channel* variables, respectively.

- The *terms* are defined by:

u, v	$::=$	x, y, z	variables from $\mathcal{X}$
		a, b, c	names from $\mathcal{N}$
		$f(t_1, \ldots, t_n)$	function application for f from $\mathcal{F}$

- The *processes* are defined by:

P, Q	$::=$	0	the process that does nothing
		$in(u, x).P$	the input on channel u is bound to x
		$out(u, t).P$	the output on channel u is t
		$let\ x = t\ in\ P$	local definition
		$if\ t_1 = t_2\ then\ P\ else\ Q$	conditional
		$\nu\, n.P$	name restriction
		$P \vert Q$	parallel composition
		$!P$	replication, infinite parallel composition

where n is a name, x is a variable, t, t_1, t_2 are terms and u stands for a channel name or a channel variable.

A π-calculus for protocols: example

For the Horn clause representation we've used the signature $\mathcal{F} = \{pk, aenc, senc, sign, pair\}$, which is a signature of constructors.

We can express the first exchange in NSPK

$A \longrightarrow B: \ \{\!|\, A, na \,|\!\}_{pk(B)}$ by

$\nu\, n.out(c, aenc(pair(skA, n), pk(skB)))$

where n, skA and skB, $c \in \mathcal{N}$ and c is a name for channels.

The second exchange in NSPK is:

$B \longrightarrow A: \ \{\!|\, na, nb \,|\!\}_{pk(A)}$

and we note that B has to obtain n_a from the initial message, which means to perform decryption and to extract the second component of the decrypted message.

In order to define B's actions, we add *destructors*, as well as equations for constructors and destructors.

Equational theory

- for asymmetric encryption the destructor is *adec* and the ecuation is $adec(aenc(x, pk(y)), y) = x$
- for the constructor *senc* the destructor is *sdec* and the equational theory for symmetric encryption is $sdec(senc(x, y), y) = x$
- for the constructors *sign*, the destructor can be *check* and the equational theory is $check(sign(x, y), pk(y)) = x$
- for the constructor *pair* the destructors are *fst* and *snd*, which extract the first and the second component.

Consequently, $\mathcal{F} = \{pk, aenc, senc, sign, pair\} \cup \{adec, sdec, check, fst, snd\}$. More operations (constructors, destructors) and equations can be considered, depending on the protocol we specify.

A π-calculus for protocols: remarks

P, Q	$::=$	0	the process that does nothing
		$in(u, x).P$	the input on channel u is bound to x
		$out(u, t).P$	the output on channel u is t
		$let\ x = t\ in\ P$	local definition
		$if\ t_1 = t_2\ then\ P\ else\ Q$	conditional
		$\nu\, n.P$	name restriction
		$P \mid Q$	parallel composition
		$!P$	replication, infinite parallel composition

- Names and variables have scopes, defined by restrictions, inputs and local definitions and one can define the free (bound) names and variables.
- The expressions defining processes are equal modulo renaming of bound names and variables.
- The formal semantics is defined by *reduction*:

$$\begin{array}{l} out(u,t).P \mid in(u,x).Q \rightarrow P \mid Q\{t/x\} \\ if\ t = t\ then\ P\ else\ Q \rightarrow P \\ if\ t_1 = t_2\ then\ P\ else\ Q \rightarrow Q \end{array}$$

  where $\{t/x\}$ is the substitution that replaces the free occurrences of x with t and $t_1 = t_2$ is not provable in the corresponding equational theory.

We refer to [4] for details.

The Denning-Sacco protocol revisited

$$\begin{array}{ll} A \longrightarrow B : & \{\!|\, \{\!|\, k \,|\!\}_{sk(A)} \,|\!\}_{pk(B)} \\ B \longrightarrow A : & \{\!|\, secret \,|\!\}_{k} \end{array}$$

In order to model our protocol we define

- a *process for each role* and
- a *process that brings them toghether*.

$$\begin{array}{ll} P_A(skA, pkB) = & \nu\, k.out(c, aenc(sign(k, skA), pkB)). \\ & in(c, x). \\ & let\ z = sdec(x, k)\ in\ 0 \end{array}$$

The Denning-Sacco protocol

$$\begin{array}{ll} A \longrightarrow B : & \{\!|\, \{\!|\, k \,|\!\}_{sk(A)} \,|\!\}_{pk(B)} \\ B \longrightarrow A : & \{\!|\, secret \,|\!\}_{k} \end{array}$$

- a *process for each role*:

$$\begin{array}{rl} P_A(skA, pkB) = & \nu\, k.out(c, aenc(sign(k, skA), pkB)). \\ & in(c, x). \\ & let\ z = sdec(x, k)\ in\ 0 \end{array}$$

$$\begin{array}{rl} P_B(skB, pkA) = & in(c, y). \\ & let\ z = adec(y, skB)\ in \\ & \quad if\ \ check(z, pkA) = xk\ \ then \\ & \qquad \nu\, s.out(c, senc(s, xk)) \end{array}$$

- a *process that brings them together*:

$$P_{DS} = \nu\, skA.\nu\, skB.P_A(skA, pk(skB)) \mid P_B(skB, pk(skA))$$

A π-calculus for protocols: NSPK

For the Needham-Scroeder Public Key Protocol (NSPK):

$$\begin{array}{ll} A \longrightarrow B: & \{\!|\, A, na \,|\!\}_{pk(B)} \\ B \longrightarrow A: & \{\!|\, na, nb \,|\!\}_{pk(A)} \\ A \longrightarrow B: & \{\!|\, nb \,|\!\}_{pk(B)} \end{array}$$

the process corresponding to the initiator can be defined by:

$$\begin{array}{rl} P_A(skA, pkB) = & \nu\, n.out(c, aenc(pair(skA, n), pkB). \\ & in(c, x). \\ & \quad let\ z = adec(x, skA)\ in \\ & \quad\quad if\ \ fst(z) = n\ then\ out(c, aenc(snd(z), pkB)\ else\ 0 \end{array}$$

The Denning-Sacco protocol continued

We can improve the representation of the protocol by:

- allowing multiple parallel executions for each role:

$$\begin{aligned} P_A(skA, pkB) = \quad & !\, \nu\, k.out(c, aenc(sign(k, skA), pkB)).\\ & in(c, x).\\ & let\ z = sdec(x, k)\ in\ 0 \end{aligned}$$

$$\begin{aligned} P_B(skB, pkA) = \quad & !\, in(c, y).\\ & let\ z = adec(y, skB)\ in\\ & \quad if\ \ check(z, pkA) = xk\ then\\ & \qquad \nu\, s.out(c, senc(s, xk)) \end{aligned}$$

- making the public keys available to a potential attacker:

$$\begin{aligned} P_{DS} = \quad & \nu\, skA.\nu\, skB.\\ & \quad let\ pkA = pk(skA)\ in\\ & \quad\ let\ pkB = pk(skB)\ in\\ & \qquad out(c, pk(skA)).out(c, pk(skB)).\\ & \qquad\ P_A(skA, pk(skB)) \mid P_B(skB, pk(skA)) \end{aligned}$$

The Denning-Sacco protocol continued

In the above representation, the (honest) participants talk only to each other. In order to allow the attacker to interfere we can modify the initiator process such that the public key of the communication partner is received through the channel c:

$$
\begin{array}{ll}
P_A(skA) = & !\, in(c, x_{pk}). \\
& \nu\, k.out(c, aenc(sign(k, skA), x_{pk}). \\
& in(c, x). \\
& let\ z = sdec(x, k)\ in\ 0
\end{array}
$$

$$
\begin{array}{ll}
P_B(skB, pkA) = & !\, in(c, y). \\
& let\ z = adec(y, skB)\ in \\
& \quad if\ check(z, pkA) = xk\ then \\
& \quad\quad \nu\, s.out(c, senc(s, xk))
\end{array}
$$

$$
\begin{array}{ll}
P_{DS} = & \nu\, skA.\nu\, skB. \\
& let\ pkA = pk(skA)\ in \\
& \quad let\ pkB = pk(skB)\ in \\
& \quad\quad out(c, pk(skA)).out(c, pk(skB)). \\
& \quad\quad\quad P_A(skA, pk(skB)) \mid P_B(skB, pk(skA))
\end{array}
$$

ProVerif

- The ProVerif language extends the π- **calculus** for protocols.
- The description of the protocol is automatically translated in a set of **Horn clauses**.
- The security property is checked by performing a **particular version of resolution**, developed for this particular setting.

`https://bblanche.gitlabpages.inria.fr/proverif/`

Thank you!

References

1 Véronique Cortier, Steve Kremer. Formal Models and Techniques for Analyzing Security Protocols: A Tutorial. Foundations and Trends in Programming Languages, Now Publishers, 2014, 1 (3), pp.117. https://hal.archives-ouvertes.fr/hal-01090874

2 Bruno Blanchet. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. Foundations and Trends® in Privacy and Security , Now publishers inc, 2016, 1 (1-2), pp.1 - 135.https://hal.inria.fr/hal-01423760/

3 Bruno Blanchet. Using Horn Clauses for Analyzing Security Protocols. In Véronique Cortier and Steve Kremer, editors, Formal Models and Techniques for Analyzing Security Protocols, volume 5 of Cryptology and Information Security Series, pages 86-111. IOS Press, March 2011.

4 Abadi, M., B. Blanchet, and C. Fournet. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. Journal of the ACM Vol. 65Issue 1February 2018 Article No.: 1pp 1–41. Available at http://arxiv.org/abs/1609. 03003v1.