JFK Protocol in Applied Pi Calculus
Proving Security

◆ “Real” protocol
  • Process-calculus specification of the actual protocol

◆ “Ideal” protocol
  • Achieves the same goal as the real protocol, but is secure by design
    • Uses unrealistic mechanisms, e.g., private channels
    • Represents the desired behavior of real protocol

◆ To prove the real protocol secure, show that no attacker can tell the difference between the real protocol and the ideal protocol
  • Proof will depend on the model of attacker observations
Example: Challenge-Response

Challenge-response protocol

\[ \begin{align*}
A & \rightarrow B & \{i\}_k \\
B & \rightarrow A & \{i+1\}_k
\end{align*} \]

This protocol is secure if it is indistinguishable from this “ideal” protocol

\[ \begin{align*}
A & \rightarrow B & \{\text{random}_1\}_k \\
B & \rightarrow A & \{\text{random}_2\}_k
\end{align*} \]
Example: Authentication

Authentication protocol

A → B \{i\}_k
B → A \{i+1\}_k
A → B "Ok"

This protocol is secure if it is indistinguishable from this "ideal" protocol

A → B \{random_1\}_k
B → A \{random_2\}_k
B → A \text{random}_1, \text{random}_2 \text{ on a magic secure channel}
A → B "Ok" if numbers on real & magic channels match
Security as Observational Equivalence

◆ Need to prove that two processes are observationally equivalent from attacker’s viewpoint

◆ Complexity-theoretic model
  • Prove that two systems cannot be distinguished by any probabilistic polynomial-time adversary
    
    [Beaver ’91, Goldwasser-Levin ’90, Micali-Rogaway ’91]

◆ Abstract process-calculus model
  • Cryptography is modeled by abstract functions
  • Prove testing equivalence between two processes
  • Proofs are easier, but it is nontrivial to show computational completeness
    
    [Abadi-Rogaway ’00]
Main Ideas

1. The adversary is the environment in which the protocol executes
   • Intuition: the network is insecure, active attacker may be the man-in-the-middle on every wire and will interact with the protocol in unpredictable ways

2. The protocol is secure if no test performed by the environment can distinguish it from the ideal functionality
   • Ideal functionality is a “magic” protocol that is secure by design and performs the same functionality as the actual protocol

By contrast, in finite-state checking the adversary is a set of explicit rules
Applied Pi Calculus: Terms

\[ M, N ::= \ x \quad \text{Variable} \]
\[ \quad | \ n \quad \text{Name} \]
\[ \quad | \ f(M_1,\ldots,M_k) \quad \text{Function application} \]

◆ Standard functions
  • pair(), encrypt(), hash(), ...

◆ Simple type system for terms
  • Integer, Key, Channel⟨Integer⟩, Channel⟨Key⟩
Applied Pi Calculus: Processes

\[ P, Q ::= \text{nil} \]  empty process
\[ \bar{u}\langle N\rangle.P \]  send term $N$ on channel $u$
\[ u(x).P \]  receive from channel $P$ and assign to $x$
\[ !P \]  replicate process $P$
\[ P|Q \]  run processes $P$ and $Q$ in parallel
\[ (\nu n)P \]  restrict name $n$ to process $P$
\[ \text{if } M = N \text{ then } P \text{ else } Q \]  conditional
Reductions

→ silent (i.e., unobservable) computation
\[ \tilde{a}\langle M\rangle.P \mid a(x).Q \rightarrow P \mid Q[M/x] \]

\[ P \text{ sends } M \text{ to } Q \text{ on internal channel } a \]

if \( M = M \) then \( P \) else \( Q \) → \( P \)

if \( M = N \) then \( P \) else \( Q \) → \( Q \)

∀ ground \( M, N \) s.t. \( M \neq N \) in eq theory

(\( \nu n \))\( \tilde{a}\langle U\rangle \)

writing to an observable channel \( c \)
\[ \tilde{a}\langle M\rangle.P \mid a(x).Q \nu y.\tilde{a}\langle y\rangle \text{ let } \{y=M\} \text{ in } (P \mid a(x).Q) \]

“free-floating” let records values known to attacker

a(U)

reading from an observable channel \( c \)
\[ \text{let } \{y=M\} \text{ in } (P \mid a(x).Q) \overset{a(y)}{\rightarrow} P \mid Q[M/y,y/x] \]
JFKr Protocol

\[ x_i = g_{dic} \]

\[ x_r = g_{dr} \]

**DH group**

\[ t_r = hash_{Kr}(x_r, N_r, N_i, IP_i) \]

\[ x_i^{dr} = x_r^{di} = x \]

\[ K_{a,e,v} = hash_x(N_i, N_r, \{a, e, v\}) \]

\[ e_i = enc_{Ke}(ID_i, ID'_r, sa_r, sig_{Ki}(N_r, N_i, x_r, x_i, g_r)) \]

\[ h_i = hash_{Ka}("i", e_i) \]

\[ e_r = enc_{Ke}(ID_r, sa_r, sig_{Kr}(x_r, N_r, x_i, N_i)) \]

\[ h_r = hash_{Ka}("r", e_r) \]
Initiator Process

[Abadi, Blanchet, Fournet ESOP ’04 --- see website]

! init^A(ID'_r,sa_i) .
\[ N_i .
\langle 1(N_i,x_i) \rangle .
\langle 2(=N_i,N_r,x_r,g_r,t_r) \rangle .
\]

$\langle N_i \rangle .

let K_{a,e,v} = \text{hash}_{x'_d}i(N_i,N_r,\{a,e,v\}) in
let s_i = \text{sig}_{K_i}(N_r,N_i,x_r,x_i,g_r) in
let e_i = \text{enc}_{K_e}(ID_i,ID'_r, sa_i,s_i) in
let h_i = \text{hash}_{K_a}("i",e_i) in
\langle 3(N_i,N_r,x_i,x_r,t_r,e_i,h_i) \rangle .
\langle 4(e_r,h_r) \rangle .

if h_r = \text{hash}_{K_a}("r",e_r) then
let (ID_r,sa_r,s_r) = \text{decrypt}_{K_e}(e_r) in
if VerifySig_{ID_r,s_r}(x_r,N_r,x_i,N_i) then
\langle ID_r,ID'_r,sa_i,sa_r,K_v \rangle

[Control] Environment starts the initiator
Create fresh nonce \( N_i \)
Send message 1 with \( N_i \) and \( x_i \)
Wait for message 2
(received \( N_i \) must be equal to previously sent \( N_i \))
[Control] Announce start of key computation
Compute shared Diffie-Hellman keys
Sign previously exchanged information
Encrypt with the newly established shared key
Compute message authentication code (MAC)
Send message 3
Wait for message 4
Check message authentication code
Decrypt with shared key
Verify signature using R’s public key
[Control] Announce completion of protocol
Responder Process for Message 1

- ! c(1(Ni,x_i)) .
- ∀Nr .
- let t_r = hash_{K_r}(x_r,N_r, Ni) in
- c〈2(N_i,N_r,x_r,g_r,t_r)〉

*Wait for message 1*
*Create fresh nonce N_r*
*Compute anti-DoS cookie*
*Send message 2*
Responder Process for Message 3

\[ c(3(N_i,N_r,x_i,x_r,t_r,e_i,h_i)) \]

Wait for message 3
Re-compute and compare anti-DoS cookie
Check for freshness to prevent replay
[Control] Announce start of key computation and allocation of session state
Compute shared Diffie-Hellman keys
Check message authentication code
Decrypt with shared key
Check if initiator is on the authorized list
Verify signature using I’s public key
[Control] Announce acceptance of message 3
Sign previously exchanged information
Encrypt with shared key
Compute message authentication code (MAC)
Send message 4

Note: active attacker may read/write communication channel c
Features of the Model

◆ Two separate processes for responder
  • To counter denial of service attacks, responder is stateless until he receives message 3
  • Responder process for message 1 must be independent from responder process for message 3

◆ Responder must keep a database of all cookies accepted after message 3 to avoid replay attacks

◆ “Control” messages on special channels announce protocol checkpoints
  • “Completed verification”, “started key computation”...
  • Not part of specification, only to help model properties
Linearization

Parallel composition of responder to message 1 and responder to message 3 is observationally indistinguishable from a single stateful process

\( R_1^A \parallel R_3^A \approx ! c(1(N_i,x_i)). \forall N_r, t_r. \overline{c}\langle 2(N_i,N_r,x_i,x_r,g_r,t_r) \rangle. \ ?c(3(=N_i,N_r,x_i,x_r=x_r,=t_r,e_i,h_i)). \)

let \( K_{a,e,v} = \text{hash}_{xidr}(N_i,N_r,\{a,e,v\}) \) in ...

(\textit{then as in } R_3^A)
Protection From Denial of Service

◆ Initiator:
  
  For any trace $\eta \xrightarrow{} S'$, for each output $\$_\langle N_i \rangle$, there are successive actions $\text{init}^A(...)$, $\_\langle 1(N_i...) \rangle$, $\_\langle 2(N_i...) \rangle$
  
  – Initiator starts his Diffie-Hellman computation only with a nonce that he previously sent to someone in message 1 and received back in message 2

◆ Responder:

  For any trace $\eta \xrightarrow{} S'$, for each output $\_\langle N_i,N_r \rangle$, there are successive actions $\_\langle 1(N_i...) \rangle$, $\langle 2(N_i,N_r...) \rangle$, $\_\langle 3(N_i,N_r...) \rangle$
  
  – Responder starts his Diffie-Hellman computation and allocates session state only after receiving the same nonce that he sent to ostensible initiator in message 2
Secrecy for Established Key

Assume $S \xrightarrow[\eta]{} S'$. For any principals $A, B$, DH exponentials $x_i, x_r$, and terms $ID'_r, sa_i$ there exists $S_3$ such that

$S' \xrightarrow{init^A(ID'_r, sa_i)} [1,2,3] S_3$

and

either $ID_A \in S^B_i$ and

$S_3 \xrightarrow{\nu K_v, accept^B(ID_a, ID'_r, sa_i, sa_r, K_v)} [4] \xrightarrow{connect^A(ID_b, ID'_r, sa_i, sa_r, K_v)} \approx \text{let } \varphi_4 \text{ in } S'$

Positive outcome: execution is not observably different from "magic" protocol in which parties agree on a new key $K_v$ without communicating

or $ID_A \notin S^B_i$ and $S_3 \approx \text{let } \varphi_3 \text{ in } S'$

Negative outcome: if initiator is not authorized, execution is not observably different from a protocol in which responder simply stops after message 3

Observable execution of $S'$ must include start of initiator and send/receive of first 3 messages

Exports $N_i, N_r, t_r$ ... to environment
Authentication for Control Actions

Assume $S \xrightarrow{\eta} S'$. The actions in $\eta$ are such that

1. For each $\text{accept}^B(ID_a, ID'_r, sa_i, sa_r, K_v)$, $ID_A \in S^B_i$ and there is distinct $\text{init}^A(ID'_r, sa_i)$

   \text{If responder announces completion of protocol, initiator is on the authorized list and previously initiated this instance of the protocol.}

2. For each $\text{connect}^A(ID_b, ID'_r, sa_i, sa_r, K_v)$, there is distinct $\text{init}^A(ID'_r, sa_i)$ and $\text{accept}^B(ID_a, ID'_r, sa_i, sa_r, K_v)$

   \text{If initiator announces completion of protocol, then he initiated this instance and responder has announced successful completion, too.}

Authentication is a correspondence property (some event happens only if another event happened previously)
Assume $S \xrightarrow[\eta]{\text{connect}^A(ID_b, ID'_r, sa_r, sa_r, K_v)} S'$. Protocol executed, and initiator announced successful completion.

1. $\xrightarrow[\eta]{\text{init}^A(ID'_r, sa_i)} [1,2,3] \xrightarrow[\text{accept}^B(ID_a, ID'_r, sa_i, sa_r, K_v)]{} [4]$ contains a series of transitions that match in the same order except possibly for arguments $x_i$ in $1^{st}$ input on $c$ and $t_r$ in $2^{nd}$ input and $3^{rd}$ output on $c$

- Responder must have announced successful completion, too
- Values received by initiator must be equal to values sent by responder
- Values received by responder must be equal to values sent by initiator (except for unauthenticated fields $x_i$ and $t_r$)

2. Let $\eta$ be $\eta'$ without these transitions.
Then (let $\varphi_4$ in $S$) $\xrightarrow[\eta']{} S'$

Technical point: variable assignment $\varphi_4$ contains all values revealed by protocol messages

Correspondence property!

See appendix B.1 of [ABF04] on how this may reveal identities of communicating parties
Detailed Proofs

- See tech report on Bruno Blanchet’s website
  
  http://www.di.ens.fr/~blanchet/crypto/jfk.html

- Some observational equivalences are proved by hand, some using automated verifier ProVerif
  - Verification scripts available on the website

- ProVerif is a general-purpose tool for security protocol analysis
  - The ProVerif paper is on the paper assignment list (hint! hint!)
Equivalence in Process Calculus

- Standard process-calculus notions of equivalence such as bisimulation are not adequate for cryptographic protocols
  - Different ciphertexts leak no information to the attacker who does not know the decryption keys
- $((\nu k)\tilde{c}\langle senc(M,k) \rangle$ and $((\nu k)\tilde{c}\langle senc(N,k) \rangle$ send different messages, but they should be treated as equivalent when proving security
  - In each case, a term is encrypted under a fresh key
  - No test by the attacker can tell these apart
Testing Equivalence

◆ Intuitively, two processes are equivalent if no environment can distinguish them

◆ A test is a process R and channel name w
  • Informally, R is the environment and w is the channel on which the outcome of the test is announced

◆ A process P passes a test (R,w) if P | R may produce an output on channel w
  • There is an interleaving of P and R that results in R being able to perform the desired test

◆ Two processes are equivalent if they pass the same tests
Advantages and Disadvantages

◆ Proving testing equivalence is hard
  • To prove security, need to quantify over all possible attacker processes and all tests they may perform
  • In applied pi calculus, can use "labeled bisimilarity"
    – Instead of arbitrary evaluation contexts, reason only about inputs and outputs (labeled transitions) on certain channels

◆ Testing equivalence is a congruence
  • Congruence = equivalence in any context
  • Can compose protocols like building blocks

◆ Equivalence is the "right" notion of security
  • Similar to definitions in complexity-theoretic crypto
Structural Equivalence

\[
\begin{align*}
P \mid \text{nil} & \equiv P \\
P \mid Q & \equiv Q \mid P \\
P \mid (Q \mid R) & \equiv (P \mid Q) \mid R \\
!P & \equiv P \mid !P \\
(\forall m)(\forall n)P & \equiv (\forall n)(\forall m)P \\
(\forall n)\text{nil} & \equiv \text{nil} \\
(\forall n)(P \mid Q) & \equiv P \mid (\forall n)Q \\
P[M/x] & \equiv P[N/x]
\end{align*}
\]

if \(n\) is not a free name in \(P\)

if \(M=N\) in the equational theory
Static Equivalence

Frames are static knowledge exported by a process to the execution environment

- Assignment of values to variables
  - \{x=M, y=\text{enc}_k(M,x), \ldots\}
- Attacker (i.e., environment) learns these values

Two frames $\phi$ and $\psi$ are statically equivalent if they map the same variables to equal values

- $\text{Dom}(\phi) = \text{Dom}(\psi)$ and $\forall$ terms $M, N$ $(M=N)\phi \iff (M=N)\psi$

Two processes are \textit{statically equivalent} if they export the same knowledge to the environment

- $A \approx_s B$ if their frames are statically equivalent
Labeled Bisimilarity

Labeled bisimilarity is the largest symmetric relation on closed processes s.t. \( A \sim B \) implies:

1. \( A \approx_s B \)
2. If \( A \rightarrow A' \), then \( B \rightarrow^* B' \) and \( A' \sim B' \) for some \( B' \)
3. If \( A \overset{\alpha}{\rightarrow} A' \) and \( \text{freevars}(\alpha) \subseteq \text{dom}(A) \) and \( \text{boundnames}(\alpha) \cap \text{freenames}(B) = \emptyset \), then \( B \rightarrow^* \overset{\alpha}{\rightarrow} \rightarrow^* B' \) and \( A' \sim B' \) for some \( B' \)

Why labeled bisimilarity?

- Congruence: \( \forall \) context \( C[\_], A \approx_i B \) implies \( C[A] \approx_i C[B] \)
- Easier to check than direct observational equivalence: only care about steps that export values to environment