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

  $A \rightarrow B \quad \{i\}_k$

  $B \rightarrow A \quad \{i+1\}_k$

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

  $A \rightarrow B \quad \{random_1\}_k$

  $B \rightarrow A \quad \{random_2\}_k$

Example: Authentication

- Authentication protocol

  $A \to B$ $\{i\}_k$

  $B \to A$ $\{i+1\}_k$

  $A \to B$ "Ok"

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

  $A \to B$ $\{random_1\}_k$

  $B \to A$ $\{random_2\}_k$

  $B \to A$ $random_1, random_2$ on a magic secure channel

  $A \to B$ "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

By contrast, in finite-state checking the adversary is a set of explicit rules

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

Applied Pi Calculus: Terms

$M, N ::=$	x	*Variable*
	$\mid\ n$	*Name*
	$\mid\ f(M_1,\ldots,M_k)$	*Function application*

- Standard functions
  - pair(), encrypt(), hash(), …
- Simple type system for terms
  - Integer, Key, Channel⟨Integer⟩, Channel⟨Key⟩

Applied Pi Calculus: Processes

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

Reductions

$\rightarrow$ silent (i.e., unobservable) computation

$\bar{a}\langle M\rangle.P \mid a(x).Q \;\rightarrow\; P \mid Q[M/x]$ *P sends M to Q on <u>internal</u> channel a*

$\text{if } M = M \text{ then } P \text{ else } Q \;\rightarrow\; P$

$\text{if } M = N \text{ then } P \text{ else } Q \;\rightarrow\; Q$ *∀ ground M, N s.t. M ≠ N in eq theory*

$\xrightarrow{(\nu n)\bar{a}\langle U\rangle}$ writing to an observable channel c

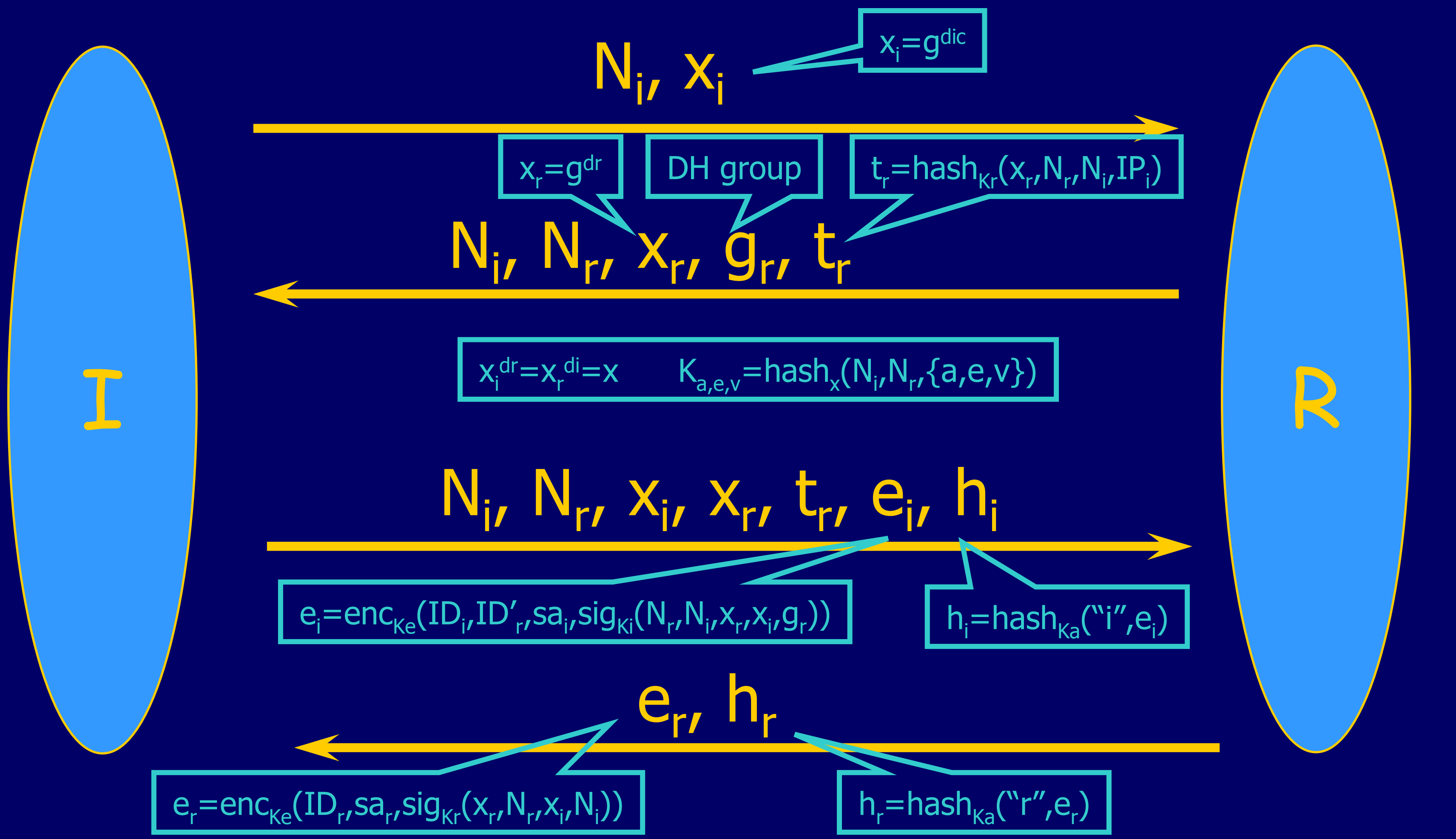
$\bar{a}\langle M\rangle.P \mid a(x).Q \;\xrightarrow{\nu y.\bar{a}\langle y\rangle}\; \text{let } \{y=M\} \text{ in } (P \mid a(x).Q)$

"free-floating" let records values known to attacker

$\xrightarrow{a(U)}$ reading from an observable channel c

$\text{let } \{y=M\} \text{ in } (P \mid a(x).Q) \;\xrightarrow{a(y)}\; P \mid Q[M/y,y/x]$

JFKr Protocol



Initiator Process

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

$! \; init^A(ID'_r, sa_i)$.	[Control] *Environment starts the initiator*
νN_i .	*Create fresh nonce N_i*
$\overline{c}\langle 1(N_i, x_i)\rangle$.	*Send message 1 with N_i and x_i*
$c(2(=N_i, N_r, x_r, g_r, t_r))$.	*Wait for message 2*
	(received N_i must be equal to previously sent N_i)
$\overline{\$}\langle N_i\rangle$.	[Control] *Annonce start of key computation*
let $K_{a,e,v}=hash_{x_r d_i}(N_i, N_r, \{a,e,v\})$ in	*Compute shared Diffie-Hellman keys*
let $s_i=sig_{K_i}(N_r, N_i, x_r, x_i, g_r)$ in	*Sign previously exchanged information*
let $e_i=enc_{K_e}(ID_i, ID'_r, sa_i, s_i)$ in	*Encrypt with the newly established shared key*
let $h_i=hash_{K_a}(\text{"i"}, e_i)$ in	*Compute message authentication code (MAC)*
$\overline{c}\langle 3(N_i, N_r, x_i, x_r, t_r, e_i, h_i)\rangle$.	*Send message 3*
$c(4(e_r, h_r))$.	*Wait for message 4*
if $h_r=hash_{K_a}(\text{"r"}, e_r)$ then	*Check message authentication code*
let $(ID_r, sa_r, s_r)=decrypt_{K_e}(e_r)$ in	*Decrypt with shared key*
if $VerifySig_{ID_r, s_r}(x_r, N_r, x_i, N_i)$ then	*Verify signature using R's public key*
$\overline{connect}^A\langle ID_r, ID'_r, sa_i, sa_r, K_v\rangle$	[Control] *Announce completion of protocol*

Responder Process for Message 1

$! \; c(1(N_i, x_i))\ .$	*Wait for message 1*
$\nu N_r\ .$	*Create fresh nonce N_r*
$\text{let } t_r = \text{hash}_{K_r}(x_r, N_r, N_i) \text{ in}$	*Compute anti-DoS cookie*
$\overline{c}\langle 2(N_i, N_r, x_r, g_r, t_r)\rangle$	*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*
if $t_r=hash_{K_r}(x_r,N_r, N_i)$ then	*Re-compute and compare anti-DoS cookie*
if t_r hasn't been accepted before then	*Check for freshness to prevent replay*
$\overline{\$}\langle N_i,N_r\rangle$.	[Control] *Announce start of key computation and allocation of session state*
let $K_{a,e,v}=hash_{x_i d_r}(N_i,N_r,\{a,e,v\})$ in	*Compute shared Diffie-Hellman keys*
if $h_i=hash_{K_a}(\text{"i"},e_i)$ in	*Check message authentication code*
let $(ID_i, ID'_r,sa_i,s_i)=decrypt_{K_e}(e_i)$ in	*Decrypt with shared key*
if $ID_i \in S_i^B$ then	*Check if initiator is on the authorized list*
if $VerifySig_{ID_i,s_i}(N_i,N_r,x_i,x_r,g_r)$ then	*Verify signature using I's public key*
$\overline{accept}^A\langle ID_i,ID'_r,sa_i,sa_r,K_v\rangle$.	[Control] *Announce acceptance of message 3*
let $s_r=sig_{K_r}(x_r,N_r,x_i,N_i))$ in	*Sign previously exchanged information*
let $e_r=enc_{K_e}(ID_r,sa_r,s_r)$ in	*Encrypt with shared key*
let $h_r=hash_{K_a}(\text{"r"},e_r)$ in	*Compute message authentication code (MAC)*
$\overline{c}\langle 4(e_r,h_r)\rangle$	*Send message 4*

<u>Note</u>: 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"…
  - <u>Not part of specification</u>, 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

$$R_1^A \mid R_3^A \approx\ !\, c(1(N_i,x_i)).\ \nu N_r,t_r.$$
$$\overline{c}\langle 2(N_i,N_r,x_r,g_r,t_r)\rangle.$$
$$?c(3(=N_i,=N_r,x_i,=x_r,=t_r,e_i,h_i)).$$
$$\text{let } K_{a,e,v}=hash_{x_i dr}(N_i,N_r,\{a,e,v\}) \text{ in } \ldots$$
$$\textit{(then as in } R_3^A\textit{)}$$

Anti-DoS cookie must appear new and random to external observer

This is the actual process executed by responder

This is what the responder's behavior must look like to any external observer

Protection From Denial of Service

- Initiator:

  For any trace $S \xrightarrow{\eta} S'$, for each output $\overline{\$}\langle N_i \rangle$, there are successive actions $init^A(\ldots)$, $\overline{c}\langle 1(N_i\ldots)\rangle$, $c(2(N_i\ldots))$

  – 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 $S \xrightarrow{\eta} S'$, for each output $\overline{\$}\langle N_i,N_r \rangle$, there are successive actions $c(1(N_i\ldots))$, $\overline{c}\langle 2(N_i,N_r\ldots)\rangle$, $c(3(N_i,N_r\ldots))$

  – 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

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

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

and

either $ID_A \in S^B_i$ and

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

***Positive outcome**: execution is not observably different from "magic" protocol in which parties agree on a new key K_v <u>without</u> communicating*

Exports N_i,N_r,t_r … to environment

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*

Authentication for Control Actions

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

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

   If responder announces completion of protocol, initiator is on the authorized list and previously initiated this instance of the protocol

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

   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)

Authentication for Complete Sessions

Assume $S \xrightarrow{\eta} \xrightarrow{\overline{\text{connect}}^{A}(ID_b, ID'_r, sa_i, sa_r, K_v)} S'$.

Protocol executed, and initiator announced successful completion

1. $\xrightarrow{\eta}$ contains a series of transitions that match

   $\xrightarrow{\text{init}^{A}(ID'_r, sa_i)} \xrightarrow{[1,2,3]} \xrightarrow{\overline{\text{accept}}^{B}(ID_a, ID'_r, sa_i, sa_r, K_v)} \xrightarrow{[4]}$

   in the same order except possibly for arguments

   x_i in 1st input on c and t_r in 2nd input and 3rd 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)*

Correspondence property!

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

2. Let η be η' without these transitions.
   Then (let φ_4 in S) $\xrightarrow{\eta'} \approx S'$

Technical point: variable assignment φ_4 contains all values revealed by protocol messages

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
- $(\nu k)\bar{c}\langle senc(M,k)\rangle$ and $(\nu k)\bar{c}\langle senc(N,k)\rangle$ 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 <u>any</u> context
  - Can compose protocols like building blocks
- Equivalence is the "right" notion of security
  - Similar to definitions in complexity-theoretic crypto

Structural Equivalence

$$P \mid nil \equiv P$$

$$P \mid Q \equiv Q \mid P$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$!P \equiv P \mid !P$$

$$(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$$

$$(\nu n)nil \equiv nil$$

$$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$$ *if n is not a free name in P*

$$P[M/x] \equiv P[N/x]$$ *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
    - $\{x=M,\ y=enc_k(M,x),\ \ldots\}$
  - Attacker (i.e., environment) learns these values
- Two frames φ and ψ are statically equivalent if they map the same variables to equal values
    - $Dom(\varphi)=Dom(\psi)$ and $\forall$ terms M, N $(M=N)\varphi$ iff $(M=N)\psi$
- Two processes are 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 $\mathcal{R}$ on closed processes s.t. A $\mathcal{R}$ B implies
  1. $A \approx_s B$
  2. If $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B'
  3. If $A \xrightarrow{\alpha} A'$ and $\mathrm{freevars}(\alpha) \subseteq \mathrm{dom}(A)$ and $\mathrm{boundnames}(\alpha) \cap \mathrm{freenames}(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B'
- Why labeled bisimilarity?
  - Congruence: $\forall$ context C[], $A \approx_l B$ implies $C[A] \approx_l C[B]$
  - Easier to check than direct observational equivalence: only care about steps that export values to environment