Crypto Protocols (the hard part…)

Today's talk includes slides from:

Bart Preneel, Andy Gordon, Jonathan Millen, and Dan Wallach

Comp527 status items

- Tutorials on using *cryptyc* by Scott Crosby
- Today: part 2, crash course in cryptography and crypto-protocols
  - Quick rehash material from last week
- Wednesday / Monday: crypto primitives
  - DES/AES/RC4/other ciphers
  - Diffie-Hellman, RSA, mathematics and such

Key establishment

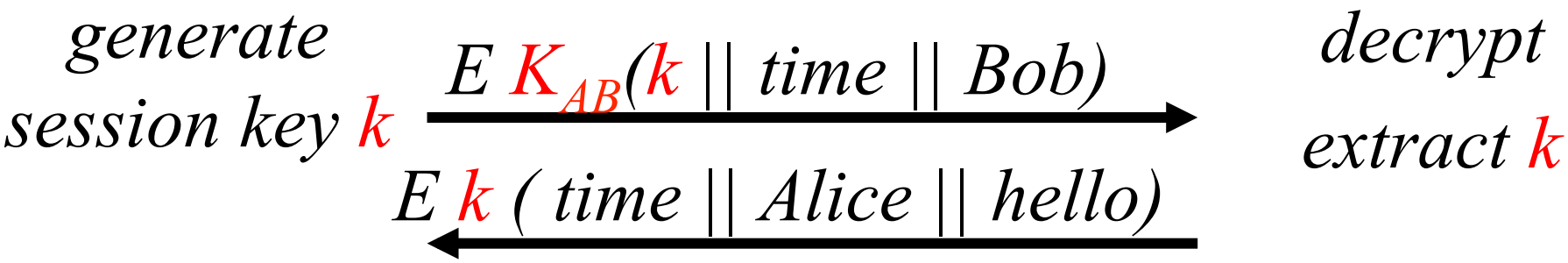
- The problem
- How to establish secret keys using secret keys?
- How to establish secret keys using public keys?
  - Diffie-Hellman and STS
- How to distribute public keys? (PKI)

Key establishment: the problem

- Cryptology makes it easier to secure information, by replacing the security of information by the security of keys
- The main problem is how to establish these keys
  - 95% of the difficulty
  - integrate with application
  - if possible transparent to end users

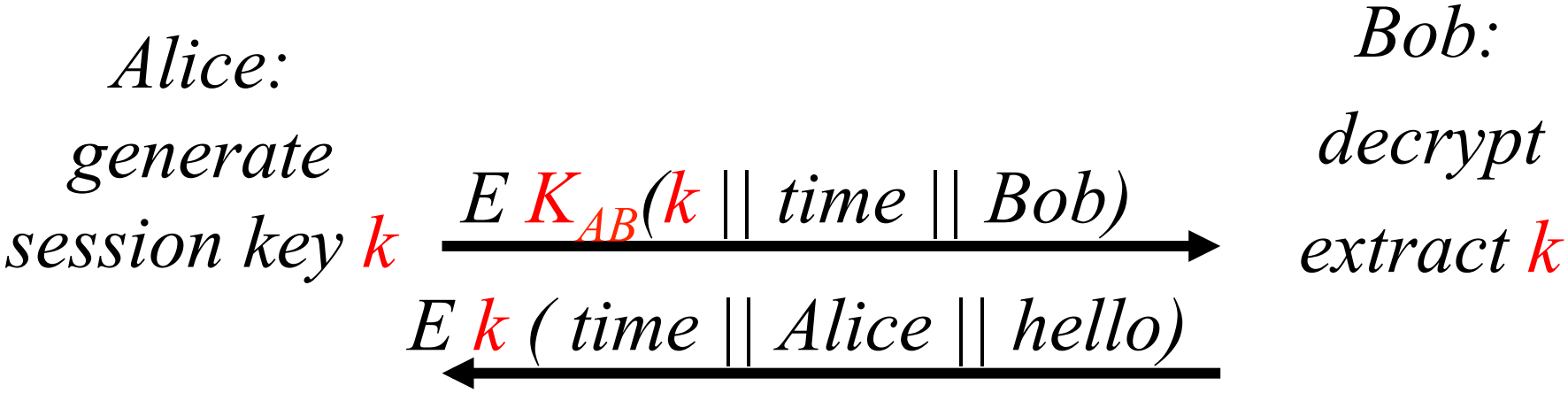
Point-to point symmetric key distribution

- Before: Alice and Bob share long term secret K_{AB}



- After: Alice and Bob share a short term key *k*
  - which they can use to protect a specific interaction
  - which can be thrown away at the end of the session

Attacks on the protocol



- Bob: proof that message is from Alice?
  - Proof that message isn't a repeat?
- Alice: proof that reply is from Bob?
  - Reply is in response to first message?

Message freshness

- Make sure a message isn't a repeat
  - Nonce handshakes (used by *cryptyc*, SSL)
    - A→B: N
    - B→A: $E_K(N)$
  - Timestamps (used by Kerberos)
  - Counters
    - A→B: $E_K(i{+}{+})$
  - Hash chains (used by S/Key)
    - A→B: $H(H(\ldots H(x)\ )) = H^{n-1}(x)$
    - B: H($H^{n-1}(x)$) → $H^n(x)$?

Folklore – Attack Terms

- *Replay*: record and later re-introduce a message or part
- *Masquerading*: pretending to be another party
  - Forge source address
- *Man-in-the-middle*: pass messages through to another session A ⇔ X ⇔ B
- *Oracle*: take advantage of unintended encryption and decryption "services" built into the protocol
- *Type confusion*: substitution of a different type of message field (e.g., key vs. nonce)

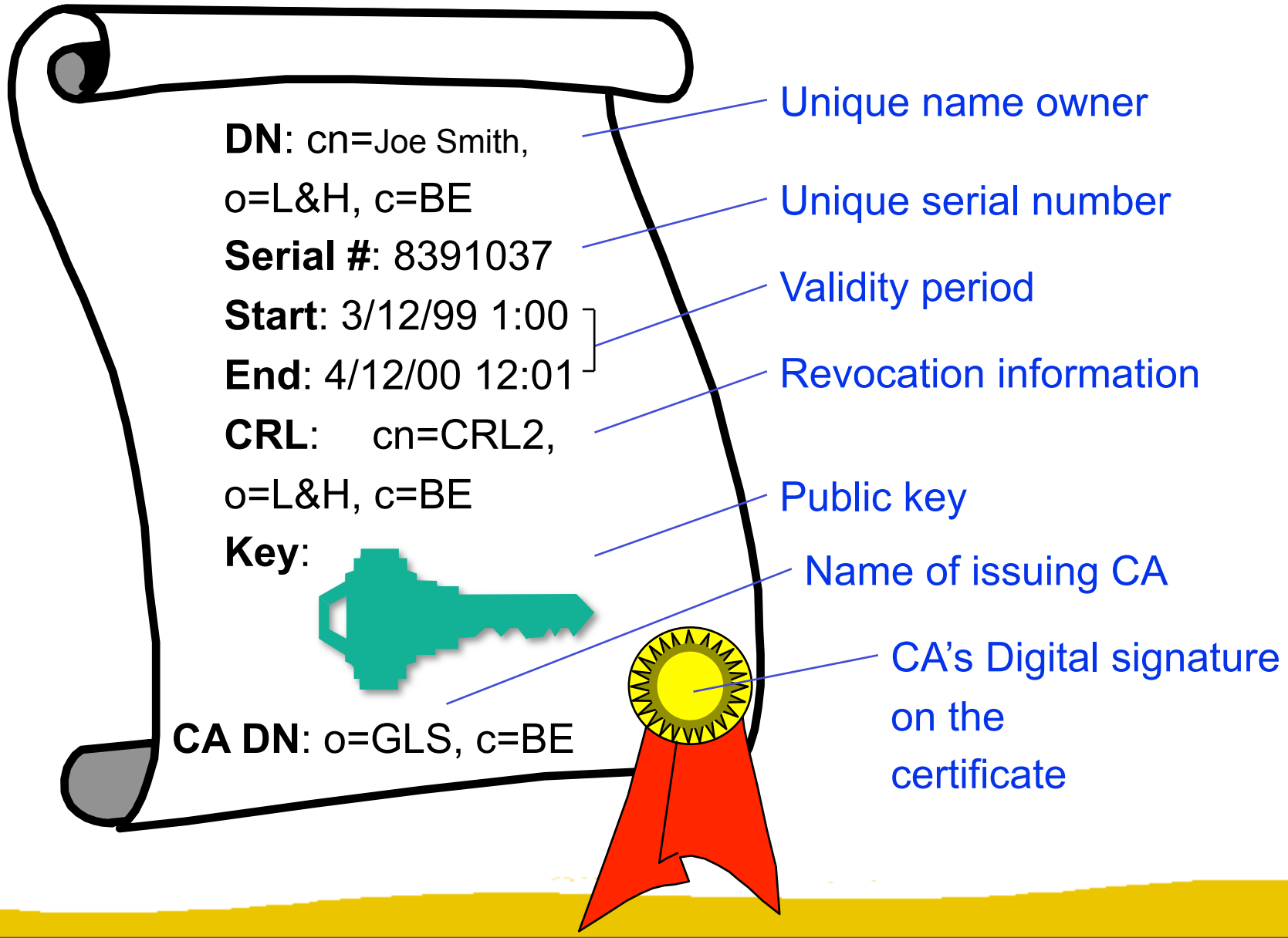
Distribution of public keys

- How do you know whose public key you have?
- Where do you get public keys?
- How do you trust public keys?
- What should you do if your private key is compromised?

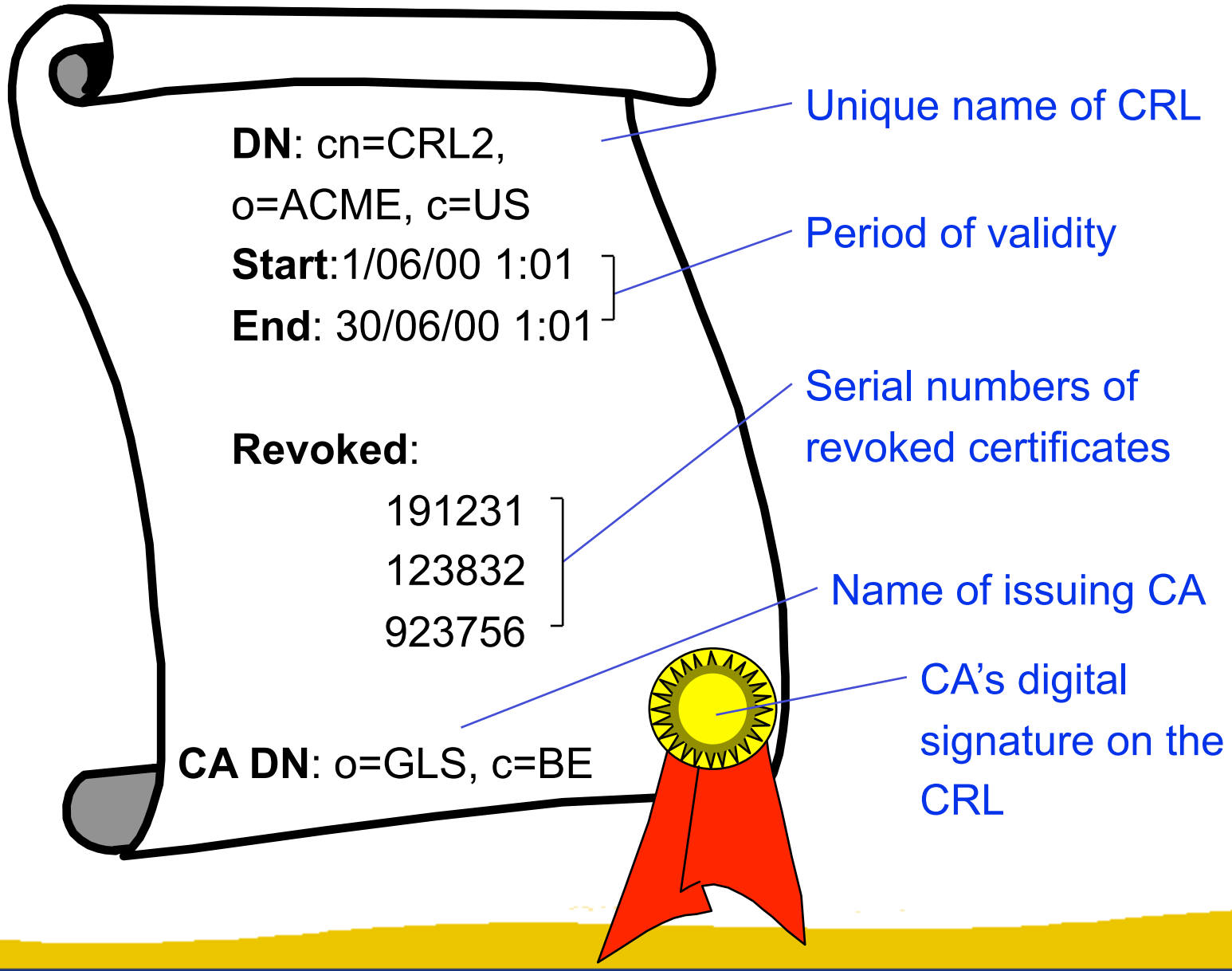
reduce protection of public key of many users to knowledge of a single public key of a Certification Authority (CA)

digital certificates &
Public Key Infrastructure (PKI)

Public Key Certificates



Certificate Revocation List

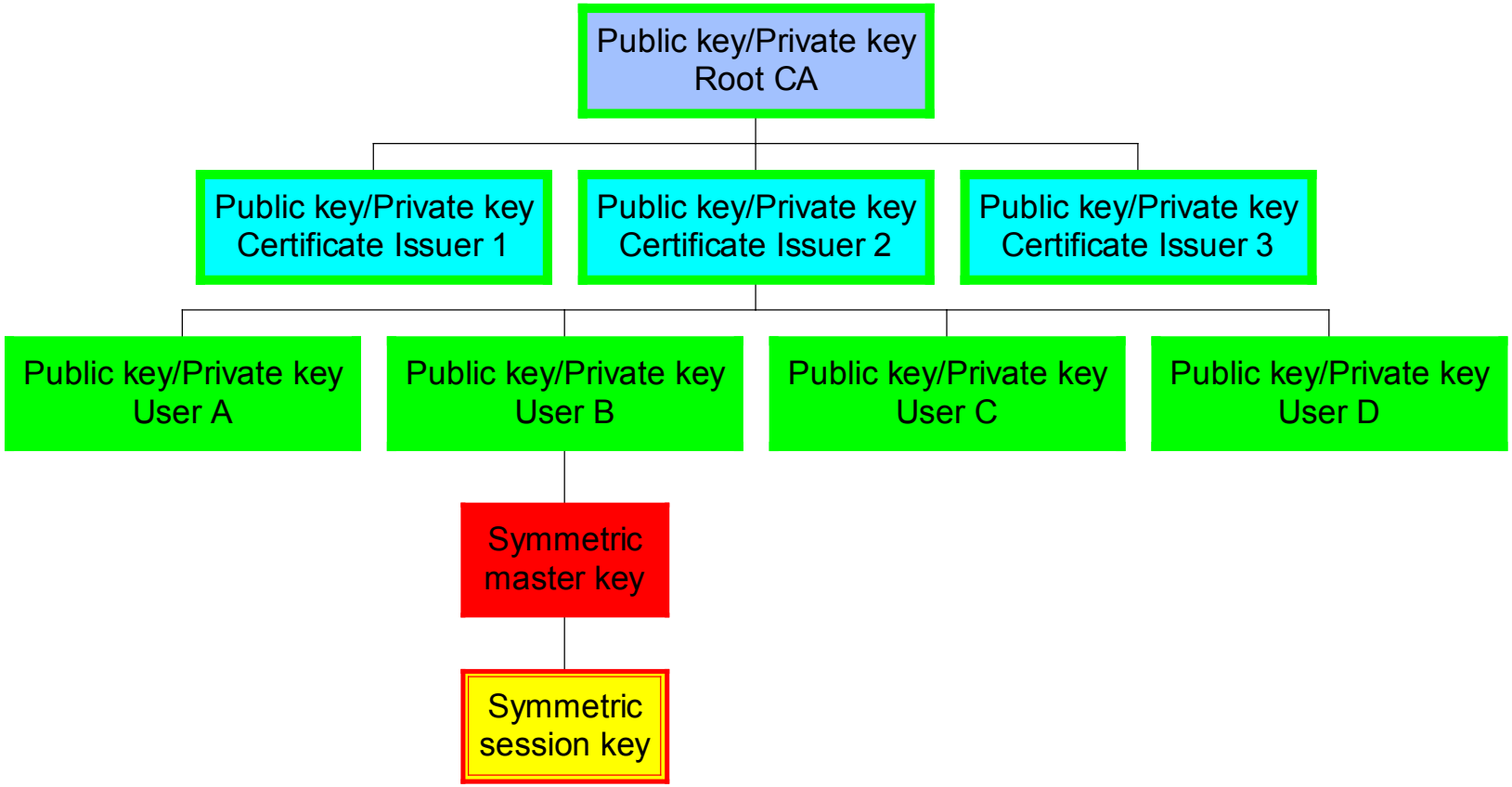


Essential PKI Components

- Certification Authority
- Revocation system
- Certificate repository (“directory”)

- Key backup and recovery system
- Support for non-repudiation
- Automatic key update
- Management of key histories
- Cross-certification
- PKI-ready application software

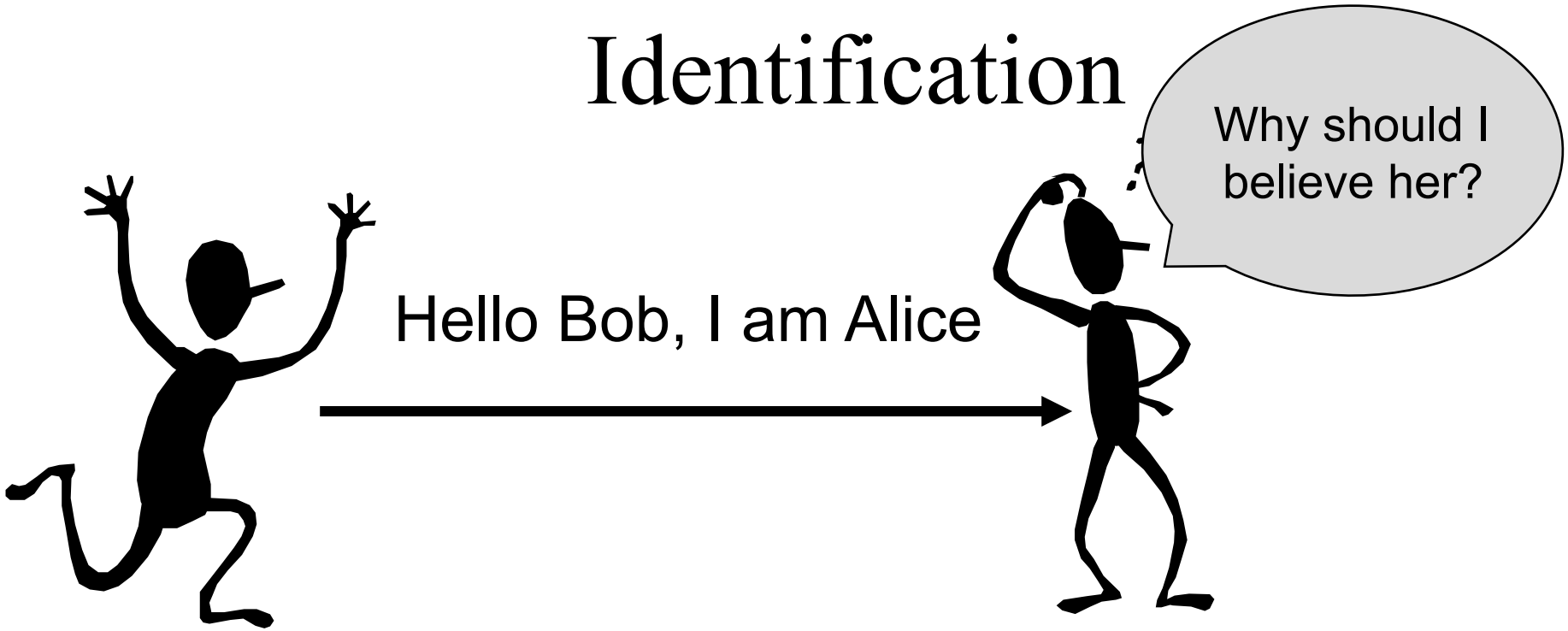
Example of a key hierarchy



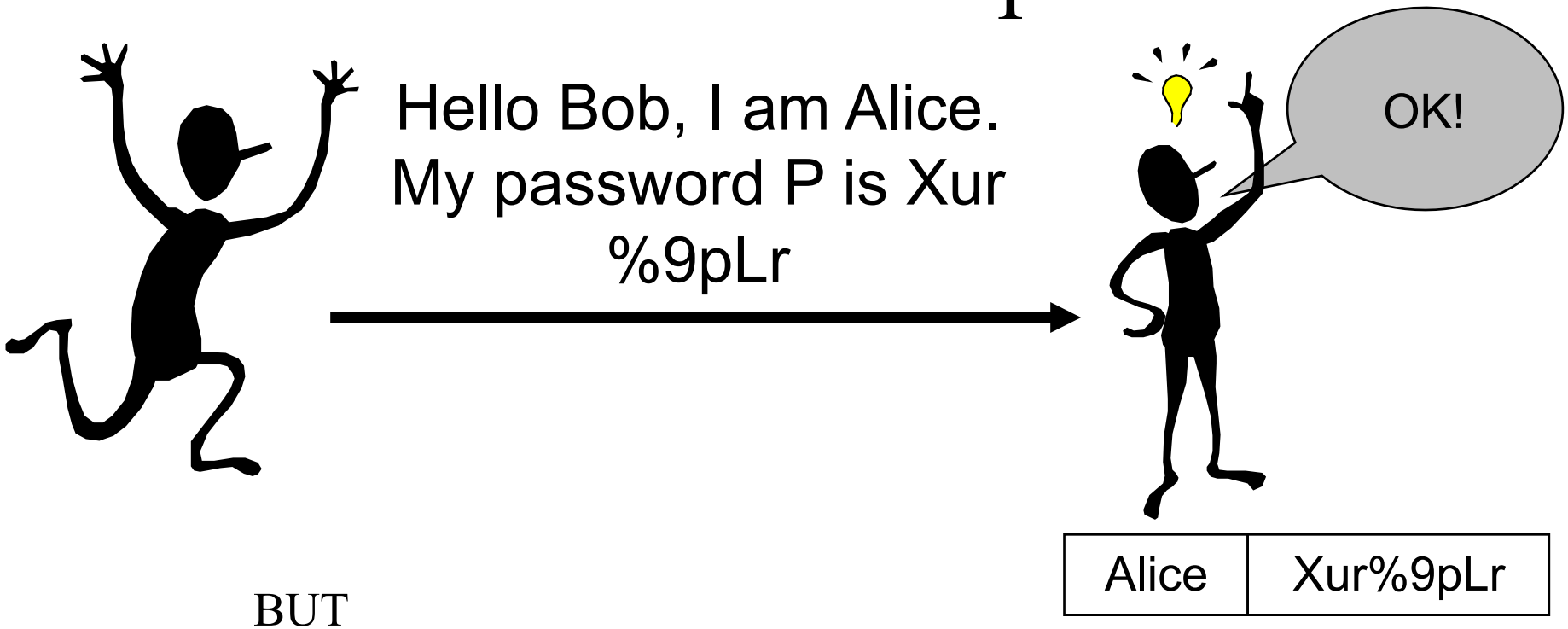
Identification

- the problem
- passwords
- challenge response with symmetric key and MAC
- challenge response with signatures

Identification



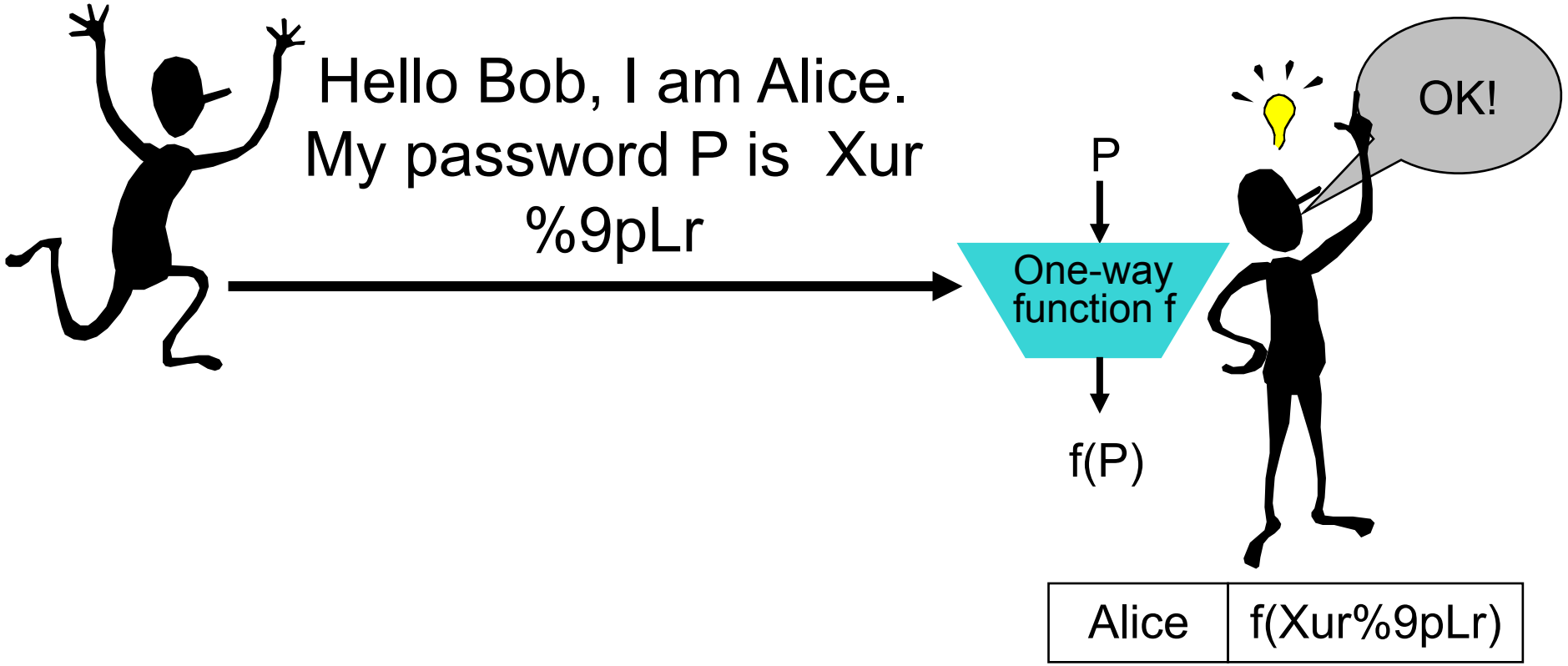
Identification with passwords



BUT

- Eve can guess the password
- Eve can listen to the channel and learn Alice's password
- Bob needs to know Alice's secret
- Bob needs to store Alice's secret in a secure way

Improved identification with passwords



Bob stores f(P) rather than Alice's secret P

- it is difficult to deduce P from f(P)

Variations on passwords

- Challenge-response (Bob must know *pw*)
  - A→B: "Hi, I'm Alice"
  - B→A: *N*
  - A→B: *Hash*(*N*, *pw*)
- Many, many recent schemes
  - Variations on Diffie-Hellman, etc.

Many different protocols

- Why not one “perfect” protocol?
  - SSL / TLS is pretty good, but…
- Many different needs / restrictions
  - Variable CPU / bandwidth resources
  - Devices without direct net connection
    - smartcards
  - Inability to have online key exchange
    - e-mail

Engineering exercise

- LiftMaster Security+ Garage door opener
  - Transmit-only
  - “Smart” door
- Threat model?
  - Passive listening attacker
- Constraints?
  - Limited memory

Receiver and Remote Control Programming

SECURITY+

NOTICE: To comply with FCC rules, adjustment or modification of this receiver and/or transmitter are prohibited, except for changing the code setting or replacing the transmitter battery. THERE ARE NO OTHER USER SERVICEABLE PARTS.

Your garage door opener receiver and remote control transmitter have been set at the factory to a matching code. The door will open when you press the remote control push button. The code between the remote control and the receiver changes with each use, randomly accessing over 100 billion new codes.

Your SECURITY+ opener will operate with:

- several SECURITY+ remote controls (with yellow indicator lights) utilizing up to 8 functions.
- one Keyless Entry System (with SECURITY+ logo)

Below are instructions for programming your opener to match any additional remotes you may purchase. See available accessories on page 38.

To Add A Remote

1. Press and *hold* the remote control push button. See Figure 1.
2. Then press and release the "Smart" (learn) button on the back panel of the opener in Figure 2. The indicator light on the panel will begin to blink and the opener light will *flash once*.
3. Release the remote push button.

Now the opener will operate when that remote control button is pressed. Test it by pressing the remote button to see that the door goes up and down.

If you release the remote control push button before the opener light flashes, the opener will not accept the code.

To Erase All Remote Control Codes

- Press and hold the "Smart" button on the opener panel until the indicator light turns off (about 6 seconds). All transmitter codes are now erased. Then follow the steps above to re-program each remote control.

Code programming instructions are also located on the opener panel.

WARNING

Children operating or playing with a garage door opener can injure themselves or others. *The garage door could close and cause serious injury or death.* Do not allow children to operate the door control(s) or remote control transmitter(s).

***A moving garage door could injure or kill someone under it.* Activate the opener only when you can see the door clearly, it is free of obstructions, and is properly adjusted.**

Figure 1

Model 91LM
SECURITY+
Single-Function Remote Control

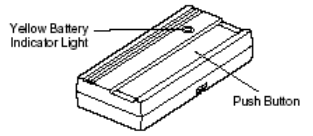
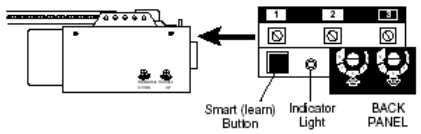


Figure 2

SECURITY+
Garage Door Opener



Garage door design space

- Remote must transmit unique messages
  - If repetition is allowed, attacker can open door
- But, nonces won't work
  - Door opener needs to remember every nonce!
- What about MAC'ed sequence numbers?
  - How to establish shared secret?
  - PKI? How does remote know door's key?
    - If all doors have same public key…

Garage door solution

- $R \rightarrow D$: K_R, $\{i{+}{+}\}$ signed by K_R
- Learning mode
  - Receive message
  - Add {key, counter} to database
- Opening mode
  - Receive message
  - Verify key, counter, signature
  - Open door

(Not sure if LiftMaster actually does this…)

Formal modelling

- You've learned a bunch of techniques…
- How to convince yourself you got it right?

Modelling crypto protocols

- Belief logics
  - Express what each message “means”
  - Derive what both parties “believe” at end
- Finite state systems
  - Model checking tools do exhaustive searches
    - Finds counter examples!
- Interactive proof systems
  - Enumerate what an attacker can learn
- Goal directed systems
  - Search space of all crypto-protocols

Comparison of modelling systems

		Little human effort	Auto attack discovery	Intuitive explanation of passes	Precise semantics	Unbounded principal size
BAN	Belief logic			✓		✓
FDR	Model checker	✓	✓		✓	
Isabelle	Interactive thm prover			✓	✓	✓
Athena	Automatic thm prover	✓	✓		✓	✓
Cryptyc	Type checker	✓		✓	✓	✓

- Beware: some columns rather subjective
- Lots of work omitted…

Protocol Vulnerabilities - Ground Rules

- Strong attacker assumption: an attacker can intercept and forge messages.
  - Possibilities: sniffers, and intrusions in firewalls or routers
  - Synonyms: attacker, intruder, spy, penetrator, enemy
- Strong encryption assumption: an attacker cannot decrypt any message without the key.
  - There are other attacks, but that's a different topic.
- Overview: Clark-Jacob survey [CJ97]

Example - Needham-Schroeder

- The Needham- Schroeder symmetric-key protocol [NS78]

  A -> S: A, B, Na

  S -> A: {Na, B, Kc, {Kc, A}Kb }Ka

  A -> B: {Kc, A}Kb

  B -> A: {Nb}Kc

  A -> B: {Nb-1}Kc

- A, B are "principals;" S is a trusted key server
- Ka, Kb are secret keys shared with S
- {X, Y}K means: X concatenated with Y, encrypted with K
- Na, Nb are "nonces;" fresh (not used before)
- Kc is a fresh connection key

Denning-Sacco Attack

- Assumes that the attacker has recorded a previous session, and compromised the connection key Kx used in that one.

  A -> B: {Kx, A}Kb *attacker replayed old message*

  B -> A: {Nb}Kx

  A -> B: {Nb-1}Kx *forged by attacker*

- B now believes he shares a fresh secret key Kx with A.
- Denning-Sacco moral: use a timestamp (calendar clock value) to detect replay of old messages.

Design Principles

- Abadi-Needham "prudent engineering practice" paraphrased
  - See [AN94]; also Anderson and Needham [AN95]
    1. Every message should say what it means.
    2. The conditions for a message to be acted on should be clearly set out.
    3. Mention the principal's name explicitly in the message if it is essential to the meaning.
    4. Be clear as to why encryption is being done.
    5. Don't assume a principal knows the content of encrypted material that is signed by that principal.
    6. Be clear on what properties you are assuming about nonces.
    7. Predictable quantities used for challenge-response should be protected from replay.

More Design Principles

8. Timestamps must take into account local clock variation and clock maintenance mechanisms.
9. A key may have been used recently, yet be old.
10. If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used.
11. The protocol designer should know which trust relations his protocol depends on.

- Good advice, but…
  - Are you sure when you have followed all of them?
  - Is the protocol guaranteed to be secure then?
  - Is it optimal and/or minimal?

Formal Methods

- Abstract modelling
- Belief logics
- State space exploration, model checking
- Inductive verification

Dolev-Yao Model

- Dolev-Yao model [DY83]
- Decidability of cascade, name-stamp ("ping-pong") protocols
  - Start with a secret M
  - Each party in turn applies a combination of operators:

    Public-key encryption Ej for any party j: M -> {M}Kj

    Public-key decryption for self, Di

    Append an identifier Ij for any party j: M -> {M, j}

    Remove an appended identifier j: Rj

    Remove any appended identifier: R
  - Example:

    A -> B: Eb(Ia(Eb(M))) = $a_1(M)$ = M' = {{M}Kb, A}Kb

    B -> A: Ea(Ib(Ea(Db(Ra(Db(M')))))) = $b_1(M')$ = Ea(Ib(Ea(M))) = M"

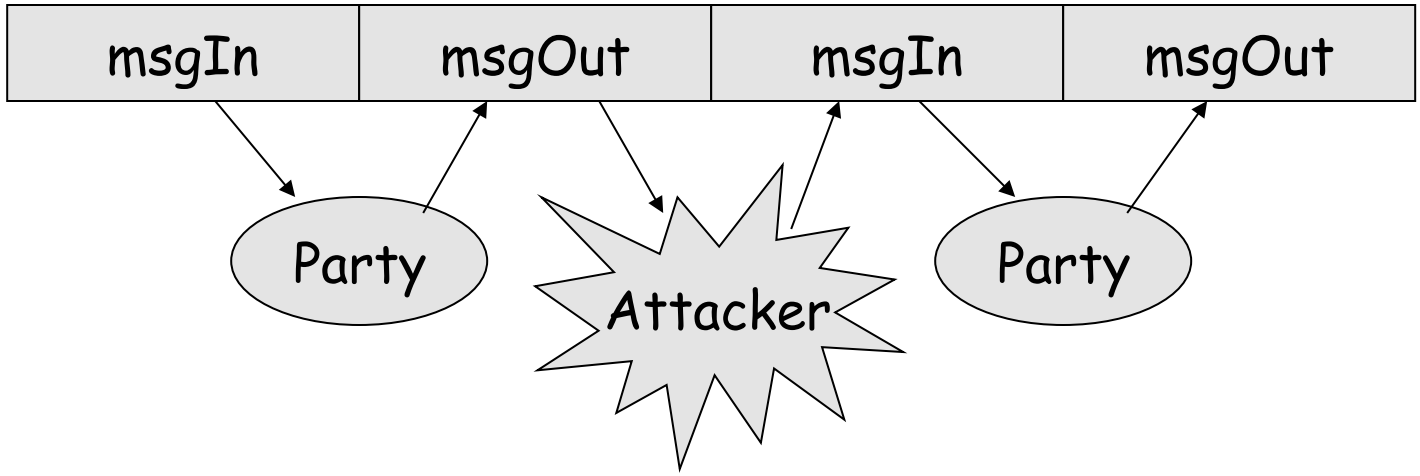
    A -> B: $a_2(M")$ ...

Dolev-Yao, cont'd

- Strong attacker and strong encryption assumptions
  - Attacker intercepts every message
  - Attacker can cause an party to apply any of its operators at any time (by starting new sessions)
  - Attacker can apply any operator except other's decryption
- Results:
  - Security decidable in polynomial time for this class
  - Security undecidable for a more general (two-field) models [EG83, HT96]
  - ***Moral: analysis is difficult because attacker can exploit interactions between two or more sessions***

Interrogator

- Prolog program [MCF87, Mil95], state-transition model
- Global state is composition of A-state, B-state, etc.
  - Party state has component per variable: A, B, Ka, etc. plus the network buffer
  - transition(state, msgIn, msgOut, nextState) defined by protocol
  - Message is a Prolog list [A, B, enc(Ka, Na)]



Interrogator, cont'd

- Attacker's transition: replace MsgOut by a MsgIn created by the attacker from known message fields
- A message field X is known by the attacker if:
  - It has appeared unencrypted in a prior message
  - It has appeared encrypted in a key known to the attacker
  - Expressed as Prolog predicate

    pKnows(X, MsgHistory, State) defined recursively
- Encryption operator rules were built in
  - Xor, exponentiation were included

Interrogator, cont'd

- Analysis technique
  - Indicate what data item X is to be kept secret
  - Define a pattern term for an INSECURE state: BadState
  - Prolog query: pKnows(X, BadState, MsgHistory)
  - Prolog will search for MsgHistory with an attack
- Features:
  - Interactive control of search
  - Output protocol diagrams
- Limitations
  - Need specific goal state to reduce search time
  - Risk of nontermination, due partly to Prolog depth-first search
  - Handles only one session per party without special encoding

NRL Protocol Analyzer

- Also in Prolog [Mea96a, Mea96b]
- State-transition rules of the form

  If:
  <conditions>
  intruderknows(<content of received message>)
  then:
  <state component assignments>
  intruderlearns(<terms>)
  EVENT:
  <named events, e.g., send message>

- State transitions may be smaller (send only, compute only)
- Reduction rules for cryptographic operators
  - Pke(privkey(X), pke(pubkey(X), Y)) => Y
  - Narrowing procedure to solve equations with these operators

NRL Protocol Analyzer, cont'd

- Features
  - Can specify message field lengths
  - Uses sequence numbers for nonces and session discrimination
  - Auxiliary analysis of "unreachable languages" can be used to prove secrecy
- Significant accomplishment: found attack on Simmons' selective broadcast protocol [Mea92].

  This protocol was Simmons' fix to an earlier protocol
- Currently in use; updated interface; ask Meadows for course

Belief Logic

- Burrows, Abadi, and Needham (BAN) Logic [BAN90a]
  - Modal logic of belief ("belief" as local knowledge)
  - Special constructs and inference rules
    e.g., P sees X (P has received X in a message)
  - Protocol messages are "idealized" into logical statements
  - Objective is to prove that both parties share common beliefs

Constructs

P bel X	P believes X
P sees X	P received X in a message
P said X	P once said X
P controls X	P has jurisdiction over X
fresh(X)	X has not been used before
P <-K-> Q	P and Q may use key K for private communication
K-> P	P has K as public key
P <-X-> Q	X is a secret shared by P and Q
{X}K	X encrypted under K
<X>Y	X combined with Y
K^{-1}	inverse key to K

(This symbolism is not quite standard)

BAN Inference Rules

- These inferences are supposed to be valid despite attacker interference.

(1) Message-meaning rules

P bel Q <-K-> P, P sees {X}K |- P bel Q said X

P bel K-> Q, P sees $\{X\}K^{-1}$ |- P bel Q said X

P bel Q <-Y-> P, P sees <X>Y |- P bel Q said X

(2) Nonce-verification

P bel fresh(X), P bel Q said X |- P bel Q bel X

(3) Jurisdiction

P bel Q controls X, P bel Q bel X |- P bel X

More BAN Rules

(4) Sees rules

P sees (X, Y) |- P sees X, P sees Y

P sees <X>Y |- P sees X

P bel Q <-K-> P, P sees {X}K |- P sees X

P bel K-> P, P sees {X}K |- P sees X

P bel K-> Q, P sees $\{X\}K^{-1}$ |- P sees X

(5) Freshness

P bel fresh(X) |- P bel fresh(X, Y) (inside encryption)

- Symmetry of <-K-> and <-X-> is implicitly used
- Conjunction is handled implicitly

  P bel (X, Y) |- P bel X and P bel Y

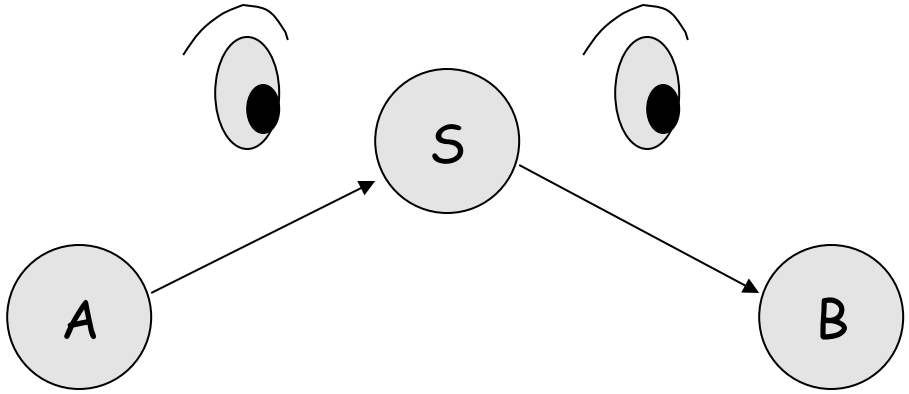
  P bel Q said (X, Y) |- P bel Q said X, P bel Q said Y

Protocol Idealization

- Convert a protocol into a collection of statements
  - Assumptions
  - Message idealizations
  - Security goals
- Message idealization conveys intent of message
  - Example: A -> B: {A, Kab}Kbs
  - Idealized: B sees {A <-Kab-> B}Kbs

- ***Note:*** *only encrypted fields are retained in the idealization.*

Example - Wide-Mouthed Frog



A -> S: A, {T, B, Kab}Kas --> (M1) S sees {T, A <-Kab-> B}Kas
S -> B: {T, A, Kab}Kbs --> (M2) B sees {T, A bel A <-Kab-> B}Kbs

(A1) P bel fresh(T), for P = A, B, S	T is a timestamp
(A2) B bel A controls A <-Kab-> B	A generates Kab
(A3) S bel A <-Kas-> S, B bel B <-Kbs-> S	Kas, Kbs are shared with S
(A4) B bel S controls A bel A <-Kab-> B	S should check this
(A5) A bel A <-Kab-> B	Justifies A said A <-Kab-> B

Analysis

- Goal: prove that B bel A <-Kab-> B.
- Proof:

B sees {T, A bel A <-Kab-> B}Kbs	M2
B bel S said (T, A bel A <-Kab-> B)	A3, rule 1
B bel fresh(T, A bel A <-Kab-> B)	A1, rule 5
B bel S bel (T, A bel A <-Kab-> B)	rule 2
B bel S bel A bel A <-Kab-> B	conjunction
B bel A bel A <-Kab-> B	A4, rule 3
B bel A <-Kab-> B	A2, Rule 3

- Exercises:
  - Prove that S bel A bel A <-Kab-> B
  - Add the message B -> A: {T}Kab (M3) and show that

    A bel B bel A <-Kab-> B

Nessett's Critique

- Awkward example in [Nes90]

  A -> B: {T, Kab}Ka^{-1} --> B sees {T, A <-Kab-> B}Ka^{-1}

- Assumptions

  (A1) B bel Ka-> A

  (A2) A bel A <-Kab-> B

  (A3) B bel fresh(T)

  (A4) B bel A controls A <-Kab-> B

- Goal: B bel A <-Kab-> B

- Proof:

B bel A said (T, A <-Kab-> B)	A1, rule 1
B bel fresh(T, A <-Kab-> B)	A3, rule 5
B bel A bel (T, A <-Kab-> B)	rule 2
B bel A <-Kab-> B	A4, rule 3

- ***Problem**: Ka is a public key, so Kab is exposed.*

Observations

- According to "Rejoinder" [BAN90b], "There is no attempt to deal with … unauthorized release of secrets"
- The logic is monotonic: if a key is believed to be good, the belief cannot be retracted
- The protocol may be inconsistent with beliefs about confidentiality of keys and other secrets
- More generally - one should analyze the protocol for consistency with its idealization
- Alternatively - devise restrictions on protocols and idealization rules that guarantee consistency

Subsequent Developments

- Discussions and semantics, e.g., [Syv91]
- More extensive logics, e.g., GNY (Gong-Needham-Yahalom) [GNY90] and SVO [SvO94]
- GNY extensions:
  - Unencrypted fields retained
  - "P possesses X" construct and possession rules
  - "not originated here" operator
  - Rationality rule: if X |- Y then P bel X |- P bel Y
  - "message extension" links fields to assertions
- Mechanization of inference, e.g, [KW96, Bra96]
  - User still does idealization
- Protocol vs. idealization problem still unsolved

Model-Checking

- Application of software tools designed for hardware CAD
  - Verification by state space exploration - exhaustive on model
- Like earlier Prolog tool approach, but
  - Forward search rather than reverse search
  - Special algorithms (BDDs, etc.)
  - A priori finite model (no unbounded recursion)
  - Fully automatic once protocol is encoded
- Practicioners:
  - Roscoe [Ros95], using FDR (the first)
  - Mitchell, et al, using Murphi [MMS97]
  - Marrero, et al, using SMV [MCJ97]
  - Denker, et al, using Maude [DMT98]
  - … and more

Model-Checking Observations

- ***Very effective*** at finding flaws, but
- No guarantee of correctnes, due to artificial finite bounds
- Setup and analysis is quick when done by experts
- Automatic translation from simple message-list format to model-checker input is possible [Low98a, Mil97]
- "Killer" example: Lowe attack on Needham-Schroeder public-key protocol, using FDR [Low96]

NSPK Protocol

- Na, Nb are nonces; PKA, PKB are public keys
- The protocol - final handshake

  A -> B: {Na, A}PKB

  B -> A: {Na, Nb}PKA

  A -> B: {Nb}PKB

- Exercise: use BAN Logic to prove

  B bel A bel A <-Nb-> B [BAN90a]

Lowe Attack on NSPK

- X is the attacker acting as a principal
- X masquerades as A for B

Session 1: A to X	Session 2: X (as A) to B
A -> X: {Na, A}PKX	
	A(X) -> B: {Na, A}PKB
	B -> A(X): {Na, Nb}PKA
X -> A: {Na, Nb}PKA	
A -> X: {Nb}PKX	
	A(X) -> B: {Nb}PKB

(Lowe's modification to fix it: B -> A: {Na, Nb, B}PKA)

Finiteness Limitation

- How many sessions must be simulated to ensure coverage?
  - Lowe attack needed two sessions
  - Example 1.3 in Dolev-Yao [DY83] needed three sessions

    A -> B: {{M}PKb, A}PKb

    B -> A: {{M}Pka, B}Pka
- No algorithmically determined bound is possible for all cases

  Because of undecidability for the model
- Possible bounds for limited classes of protocols
  - Lowe "small system" result [Low98b]: one honest agent per role, one time, if certain restrictions are satisfied:

    Encrypted fields are distinguishable

    Principal identities in every encrypted field

    No temporary secrets

    No forwarding of encrypted fields

Inductive Proofs

- Approach: like proofs of program correctness
  - Induction to prove "loop invariant"
- State-transition model, objective is security invariant
- General-purpose specification/verification system support
  - Kemmerer, using Ina Jo and ITP [Kem89] (the first)
  - Paulson, using Isabelle [Paul98] (the new wave)
  - Dutertre and Schneider, using PVS [DS97]
  - Bolignano, using Coq [Bol97]
- Can also be done manually [Sch98, THG98]
  - Contributed to better understanding of invariants
  - Much more complex than belief logic proofs
- Full guarantee of correctness (with respect to model)
  - Proofs include confidentiality

Fundamental Ideas I

- From Paulson: analz and synth
- If S is a set of message fields,

  analz(S) is the set of its components that the attacker can obtain

  synth(S) is the set of fields that the attacker can construct from it

  analz(S) is the smallest set T such that

  1. **T includes S**
  2. **if {X, Y} in T then X in T and Y in T**
  3. **if {X}K in T and K in T then X in T**

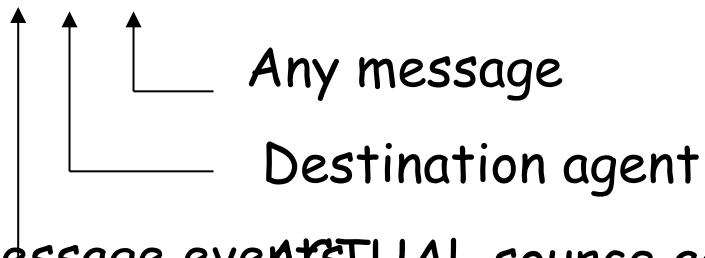
  Example: analz({{{X}K2, Y}K1, K1}) = {same, plus {{X}K2, Y}, {X}K2, Y}

  synth(S) is the smallest set T such that

  1. **T includes S**
  2. **if X in T and Y in T then {X, Y} in T**
  3. **if X in T and K in T then {X}K in T**
- The attacker can forge any message in synth(analz(S)) from S

Paulson's Modeling Approach

- Primitive message field types: Agent, Key, Nonce
- Constructed message fields: {X, Y}, Crypt K X
- Message event: Says A B X



- Trace: sequence of message events
- Protocol: set of traces
- Transition - based protocol definition of protocol P:
  - Recursive: Let T be a trace in P, then
  - (Says Spy B M) may be appended if M in synth(analz(set(T)))
  - (Says A B M) may be appended by a protocol rule

Paulson's Model, cont'd

- Typical protocol rule:

  B -> A: B, {Nb}Pka

  A -> B: A, {Na}PKb

- Formal version:

  T in P;

  Says B' A {B, Crypt (pubK A) Nonce Nb} in set T;

  Nonce Na notin used(T)

  =>

  (Says A B {A, Crypt(pubK B) Nonce Na} # T) in P

Fundamental Ideas II

- From Schneider: the *precedes* relation [Sch98] (called "authenticates" relation in an earlier conference paper)
- If S and T are sets of messages,
  - S precedes T if a message history with no occurrence of any S cannot contain an occurrence of any T
  - I.e., an observation in T implies the prior presence of an S
- Used to express authentication: a certain message received by B implies that a prior message (possibly the same message) must have been sent by *A*
- Inductive verification invariant: Suppose a history has no occurrence of S. Then it has no occurrence of T.

Fundamental Ideas III

- From Thayer, Herzog, Guttman: the *ideal* [THG98]
- Let S be the set of primitive fields that are secret by policy
  - Private or shared keys and any other shared secrets
- The ideal I[k, S] is the smallest set T such that
  1. T includes S
  2. X in T implies {X, Y} in T and {Y, X} in T (any Y)
  3. X in T and K in k implies {X}K in T
- Let k be the set of keys whose inverses are not in S.
  - Assume that keys are not computed: i.e., no key is expressible as {X, Y} or {X}Y.
- Inductive invariant for secrecy: no message is in I[k, S].
  - Sufficient because I[k, S] includes S and anything obtained by the attacker may be sent as a message.

More on the Ideal

- Is it too big? No. Necessity follows because
  - exposing {X, Y} exposes both X and Y, and
  - exposing {X}K exposes X if the inverse of K is not protected.
- The tricky part is identifying a big enough S so that the invariant holds.
- Relation to analz and synth:
  - Theorem (JKM): Let P be the complement of I[k, S] (k as above). Then synth(analz(P)) = P.
  - Proof: follows from
    1. analz(P) = P (by fixpoint induction)
    2. synth(P) = P (by structural induction)

Summary

- Cryptographic protocol verification is based on models where
  - Encryption is perfect (strong encryption)
  - The attacker intercepts all messages (strong attacker)
  - Security is undecidable in general, primarily because the number of sessions is unbounded.
- Belief logic analysis:
  - Requires "idealization" of the protocol
  - Does not address confidentiality
  - Can be peformed easily, manually or with automated support
- State-exploration approaches
  - Use model-checking tools
  - Are effective for finding flaws automatically
  - Are limited by finiteness

Summary, cont'd

- Inductive proofs
  - Can prove correctness
  - Require substantial effort
  - Can be done manually, but preferably with verification tools
- Protocol security verification is still a research area
  - But experts can do it fairly routinely
- "Real" protocols are difficult to analyze for practical reasons
  - Specifications are not precise
  - They use operators with more complex properties than simple abstract encryption
  - Flow of control is more complex - protocols negotiate alternative encryption algorithms and other parameters
  - Messages have many fields not relevant to provable security