0x1A Great Papers in Computer Security

Vitaly Shmatikov

http://www.cs.utexas.edu/~shmat/courses/cs380s/

Cryptographic Protocols

- Use cryptography to achieve some higher-level security objective
  - Authentication, confidentiality, integrity, key distribution or establishment…
- Examples: SSL/TLS, IPsec, Kerberos, SSH, 802.11b and 802.11i, Skype, S/MIME, hundreds of others
  - New protocols constantly proposed, standardized, implemented, and deployed

Needham-Schroeder Protocols

- Needham and Schroeder. "Using Encryption for Authentication in Large Networks of Computers" (CACM 1979)
- Initiated the field of cryptographic protocol design
  - Led to Kerberos, IPsec, SSL, and all modern protocols
- Observed the need for rigorous protocol analysis
  - "Protocols … are prone to extremely subtle errors that are unlikely to be detected in normal operation… The need for techniques to verify the correctness of such protocols is great, and we encourage those interested in such problems to consider this area."

Things Goes Wrong

- Many simple attacks against protocols have been discovered over the years
  - Even carefully designed, widely deployed protocols ...often years after the protocol has been deployed
    - Examples: SSL, SSH, 802.11b, GSM
  - Simple = attacks do not involve breaking crypto!
- Why is the problem difficult?
  - Concurrency + distributed participants + (often incorrect) use of cryptography
  - Active attackers in full control of communications
  - Implicit assumptions and goals behind protocols

M. Abadi and R. Needham

Prudent Engineering Practice for Cryptographic Protocols

(Oakland 1994)

Design Principles (1)

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

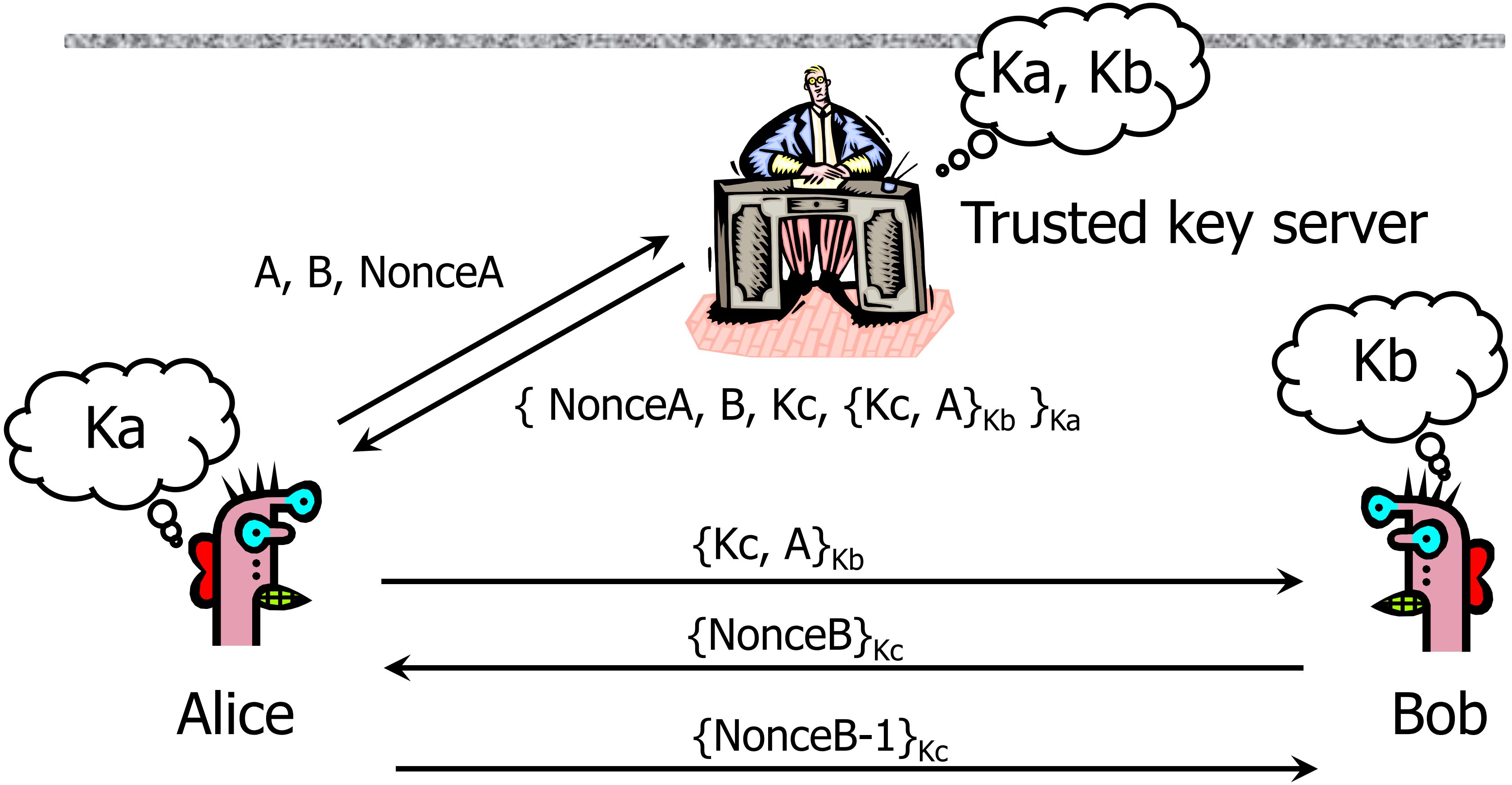
Design Principles (2)

6. Be clear on what properties you are assuming about nonces
7. Predictable quantities used for challenge-response should be protected from replay
8. Timestamps must take into account local clock variation and clock maintenance mechanisms
9. A key may have been used recently, yet be old

Design Principles (3)

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

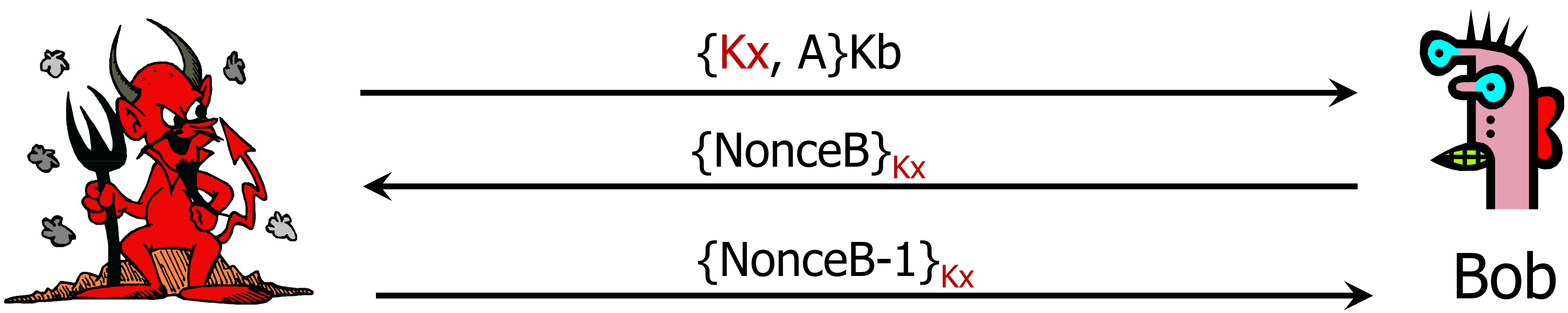
NS Symmetric-Key Protocol



- Goal: A and B establish a fresh, shared, secret key Kc with the help of a trusted key server

Denning-Sacco Attack

- Attacker recorded an old session and compromised session key Kx used in that session



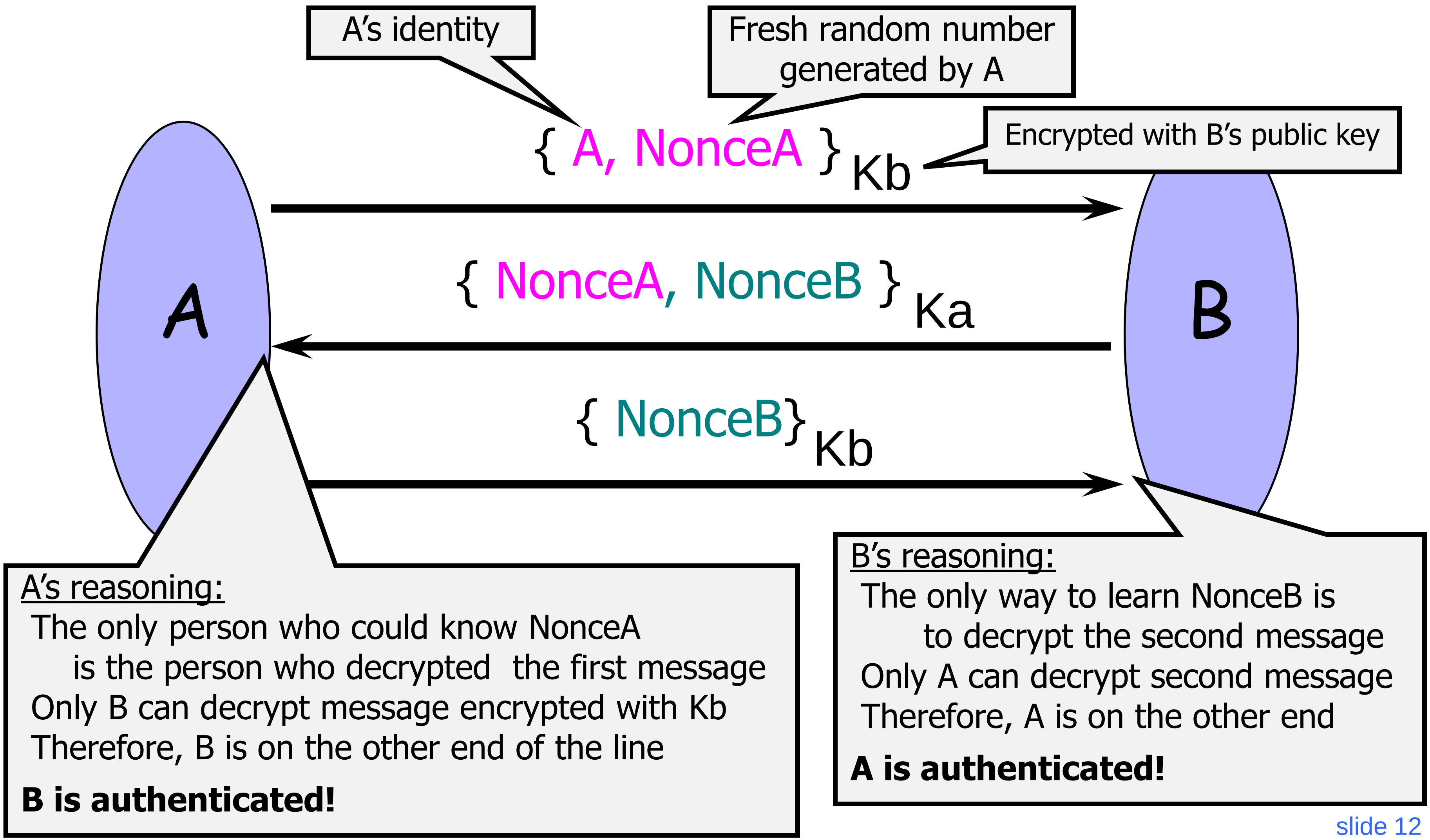
- B now believes he shares a fresh secret Kx with A
- Moral: use timestamps to detect replay of old messages

G. Lowe

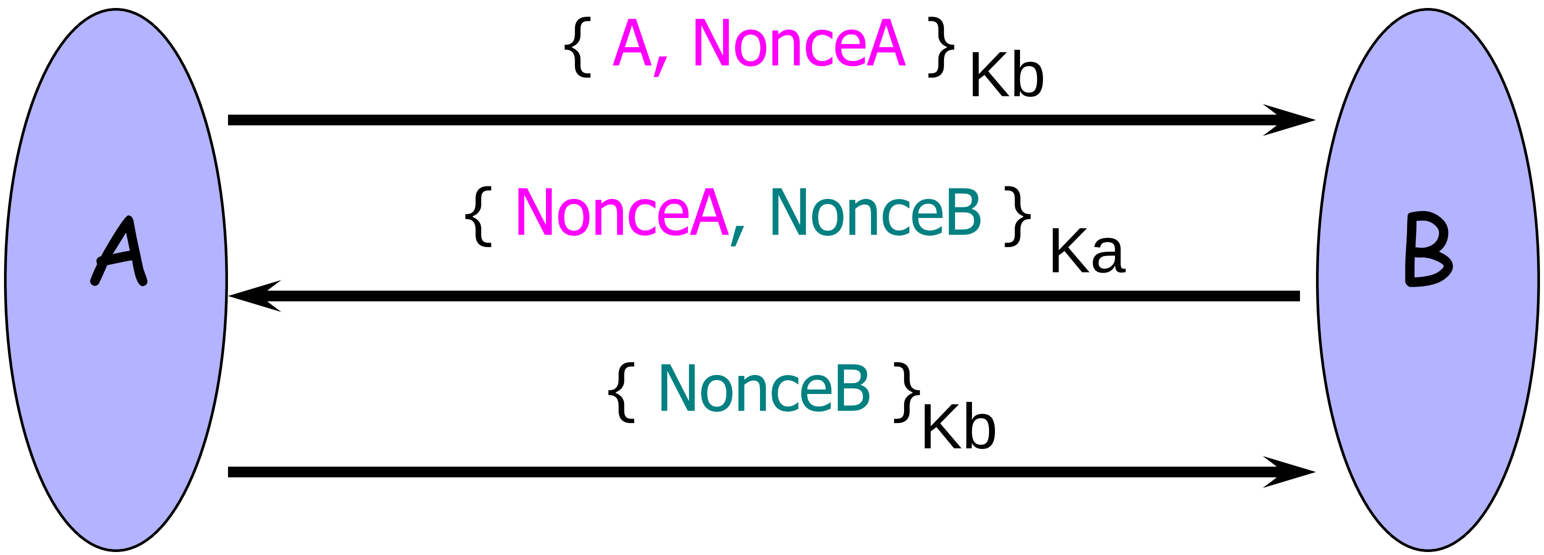
Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR

(TACAS 1996)

NS Public-Key Protocol

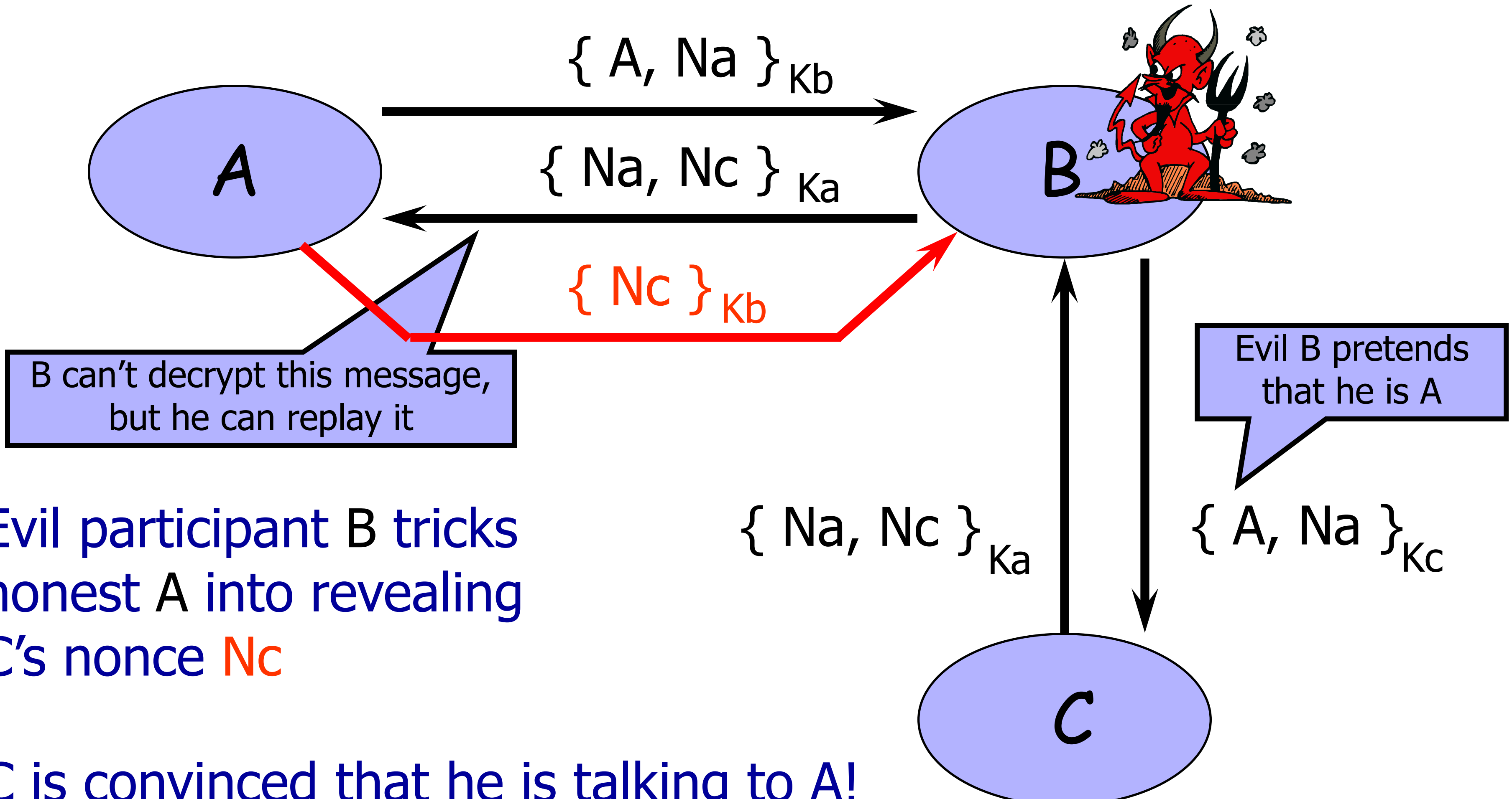


What Does This Protocol Achieve?



- Protocol aims to provide both authentication and secrecy
- After this exchange, only A and B know NonceA and NonceB ⇒ they can be used to derive a shared key

Lowe's Attack on NSPK

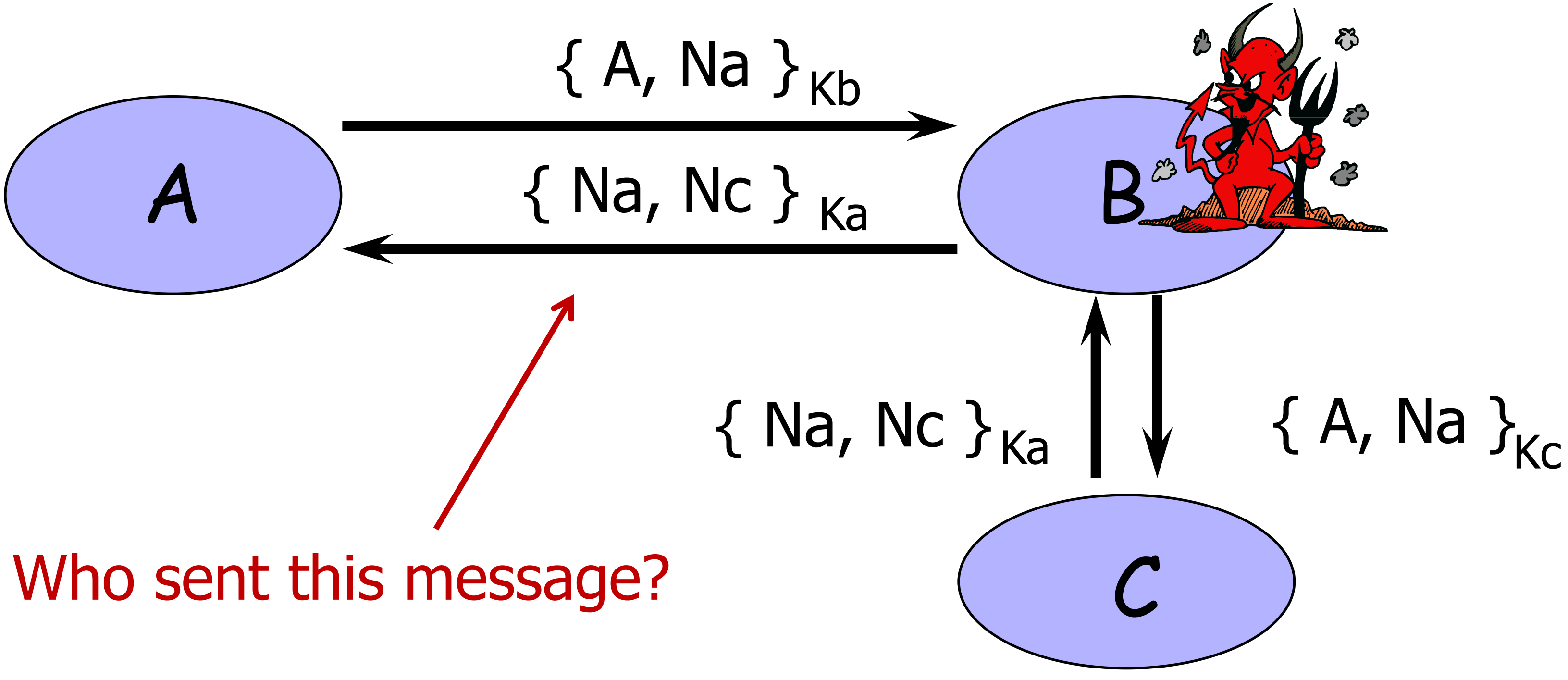


Evil participant B tricks honest A into revealing C's nonce Nc

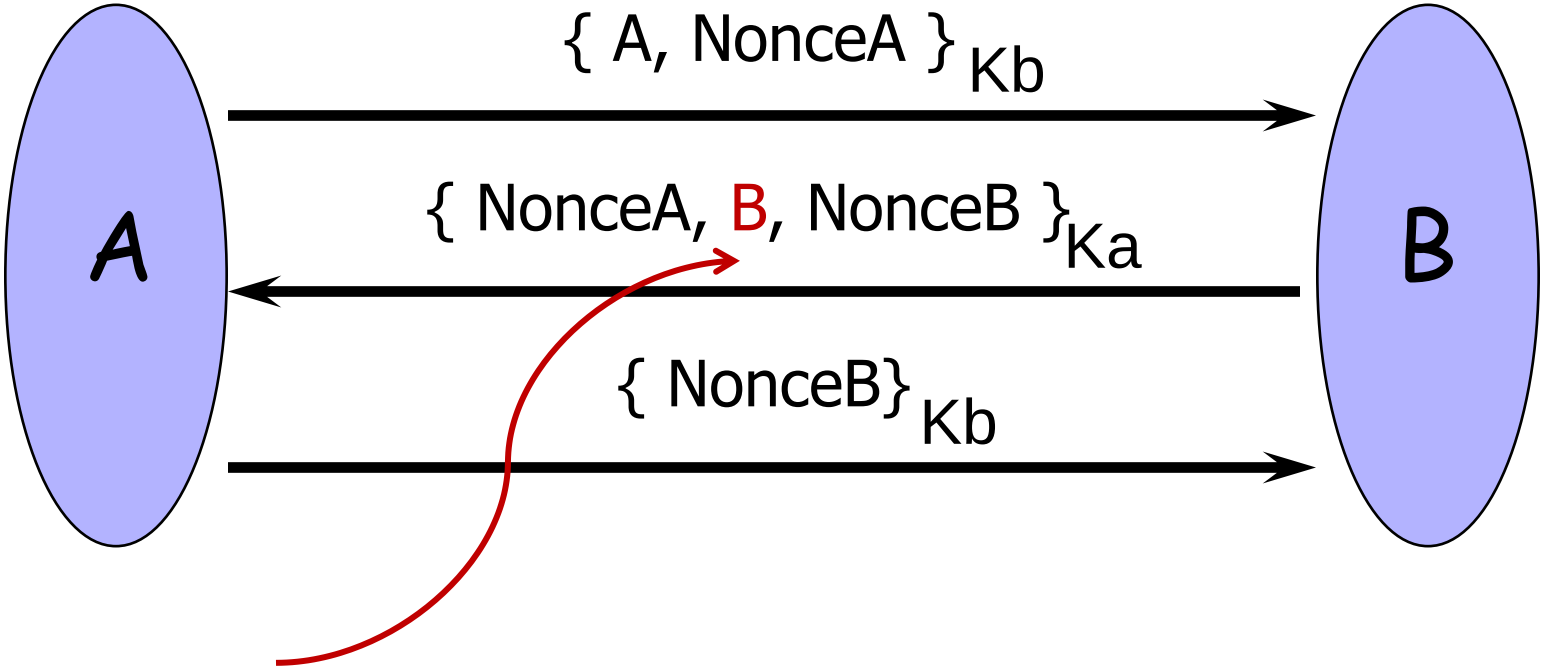
C is convinced that he is talking to A!

Abadi-Needham Principle #1

Every message should say what it means



Lowe's Fix to NSPK



Does this solve the problem? How?

Lessons of Lowe's Attack

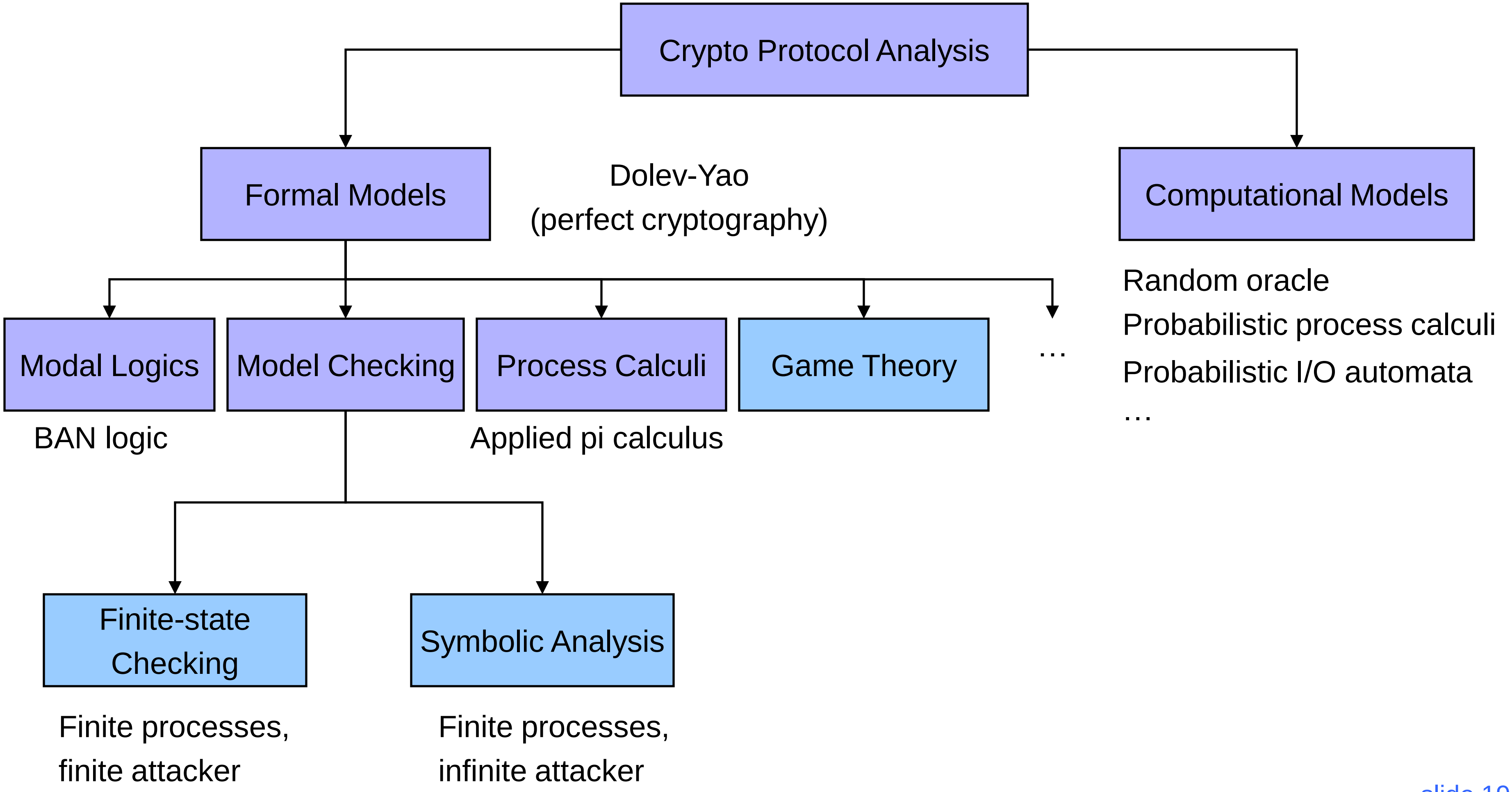
- Attacker is a legitimate protocol participant!
- Exploits participants' reasoning to fool them
  - A is correct that B must have decrypted $\{A,Na\}_{Kb}$ message, but this does not mean that the $\{Na,Nb\}_{Ka}$ message came from B
  - The attack does not rely on breaking cryptography!
- It is important to realize limitations of protocols
  - The attack requires that A willingly talk to adversary
  - In the original setting, each workstation is assumed to be well-behaved, and the protocol is correct!
- Discover attacks like this automatically?

Analyzing Security Protocols

1. Model protocol
2. Model adversary
3. Formally state security properties
4. See if properties preserved under attack

- Result: under given assumptions about the system, no attack of a certain form will destroy specified properties
  - There is no "absolute" security

Analysis Techniques



Dolev-Yao Model (1983)


- Abstract, idealized model of cryptography
  - Treat cryptographic operations as abstract data types
    - Symmetric-key decryption: decrypt($\{M\}_K$,K) = M
    - Public-key decryption: decrypt($\{M\}_{PubKey(A)}$, PrivKey(A)) = M
- Attacker is a nondeterministic process
  - Can intercept any message, decompose into parts
  - Decrypt if and only if it knows the correct key
  - Create new message from data it has observed
- Attacker cannot perform computational analysis
  - Cannot analyze actual cryptographic scheme used
  - Cannot perform statistical tests, timing attacks…

Finite-State Analysis

- Describe protocol as a finite-state system
  - State variables with initial values
  - Transition rules
  - Communication by shared variables
  - Scalable: choose system size parameters
- Specify correctness condition
- Find violations by automatic exhaustive state enumeration
  - Many tools available: FDR, Murφ, …

Rules for Protocol Participants

- Messages = abstract terms
- Participants = finite-state automata operating on terms

$A \rightarrow B \quad \{A, N_A\}_{pk(B)}$

$B \rightarrow A \quad \{N_B, N_A\}_{pk(A)}$

```
IF
  net[i].dest = B &
  net[i].encKey = B.myPubKey
THEN
  msg.nonce1:= B.myNonce;
  msg.nonce2:= net[i].nonce;
  msg.encKey:= B.keys[net[i].snd];
  net[i+1]:= msg
```

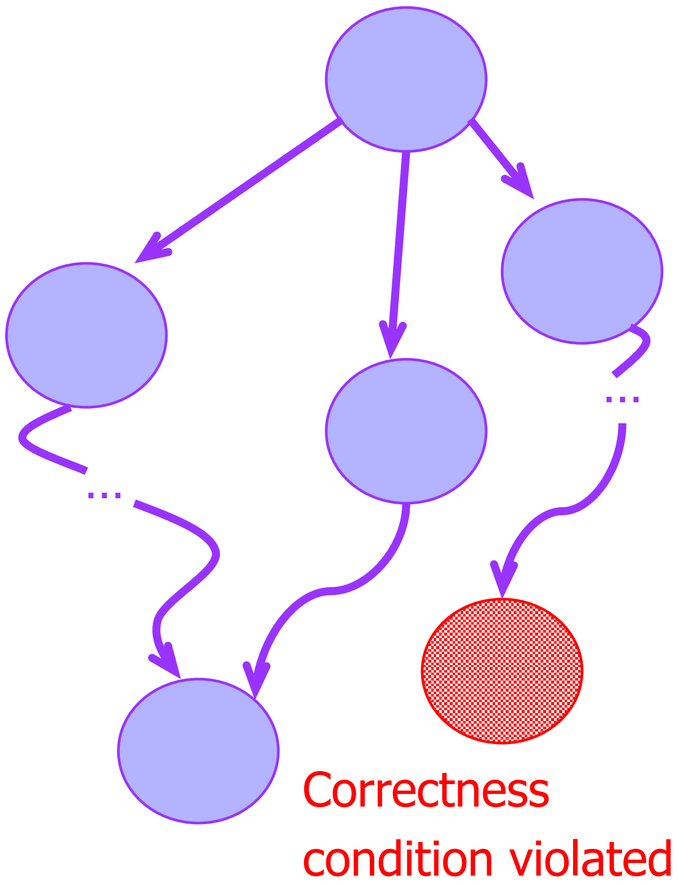
Rules for Dolev-Yao Attacker

- Read and write on the network
  - Full control over all messages exchanged by honest parties (but cannot break cryptography)
- Analyze messages
  - Decrypt if and only if correct key is known
  - Break into smaller pieces
- Construct messages
  - Concatenate known fragments
  - Encrypt with known keys

Correctness Conditions

- Specified as predicates over system variables
- Secrecy
  - ! setInclusion(B.myNonce, Attacker.KnownNonces) &
  - ! setInclusion(A.myNonce, Attacker.KnownNonces)
- Authentication
  - $\forall$ A (B.state=DONE) & (B.talkingTo=A) ->
    A.talkingTo=B

Protocol State Space



- Participant + attacker rules define a state transition graph
- Every possible execution of the protocol is a path in the graph
- Exhaustively enumerate all nodes of the graph, verify whether correctness conditions hold in every node
- If not, the path to the violating node describes the attack

Restrictions on the Model

- Two sources of infinite behavior
  - Multiple protocol runs, multiple participant roles
  - Message space or data space may be infinite
- Finite approximation
  - Assume finite number of participants
    - Example: 2 clients, 2 servers
  - Assume finite message space
    - Represent random numbers by r1, r2, r3, …
    - Do not allow encrypt(encrypt(encrypt(…)))

This restriction is necessary for decidability

This is restriction is **not** necessary (symbolic analysis!)

Tradeoffs

- Finite models are abstract and greatly simplified
  - Components modeled as finite-state machines
  - Cryptographic functions modeled as abstract data types
  - Security property stated as unreachability of “bad” state
- They are tractable…
  - Lots of verification methods, many automated
- …but not necessarily sound
  - Proofs in the abstract model are subject to simplifying assumptions which ignore some of attacker’s capabilities
- Attack in the finite model implies actual attack