CS 6431

Security Protocols

Vitaly Shmatikov

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

Design Principles (1)

[Abadi and Needham. "Prudent Engineering Practice for Cryptographic Protocols ". Oakland 1994]

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)

[Abadi and Needham]

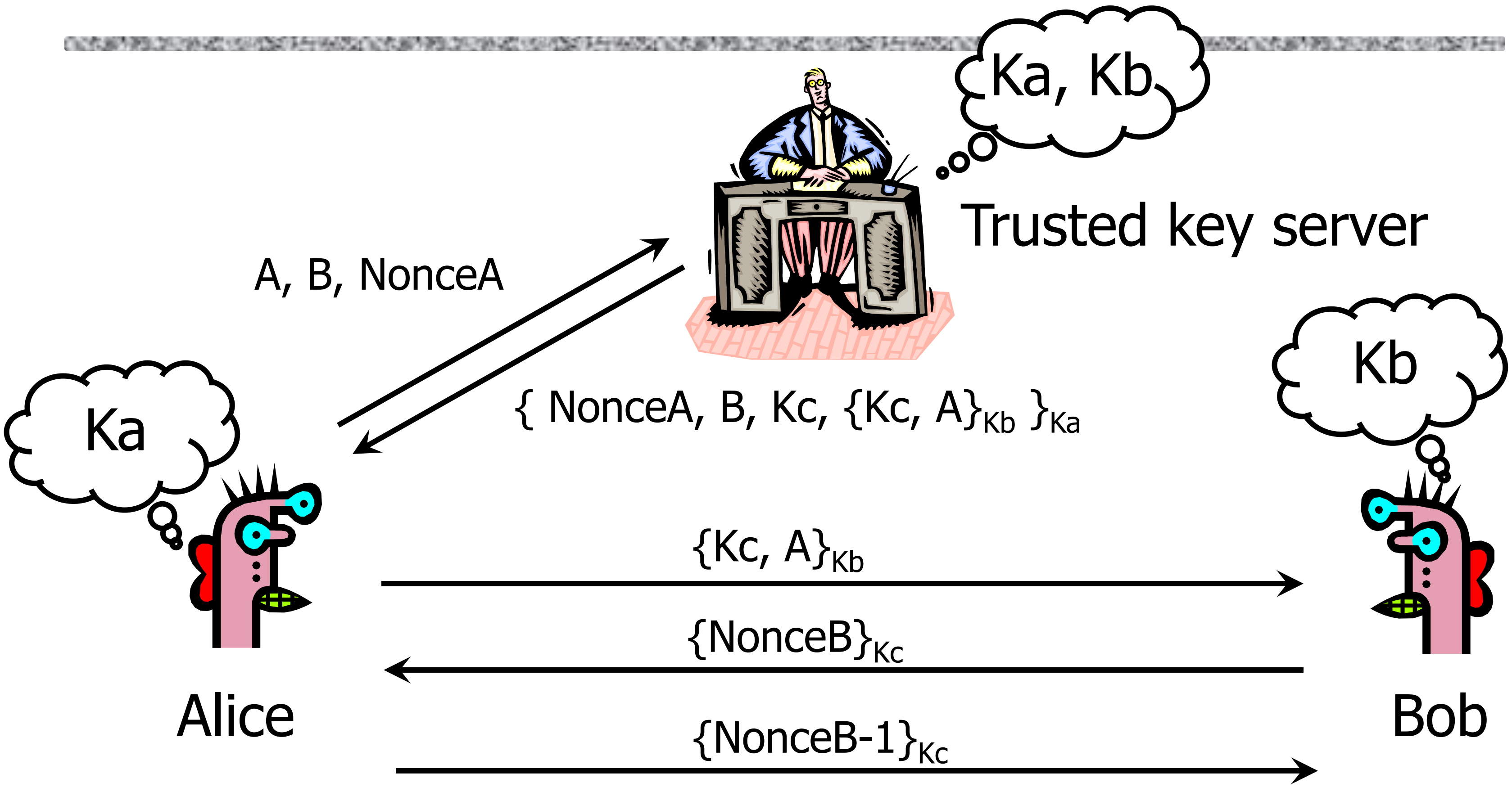
6. Be clear on what properties you are assuming about nonces
7. Predictable quantities used for challenge-response should be protected from replay
8. Timestamps must take into account local clock variation and clock maintenance mechanisms
9. A key may have been used recently, yet be old

Design Principles (3)

[Abadi and Needham]

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

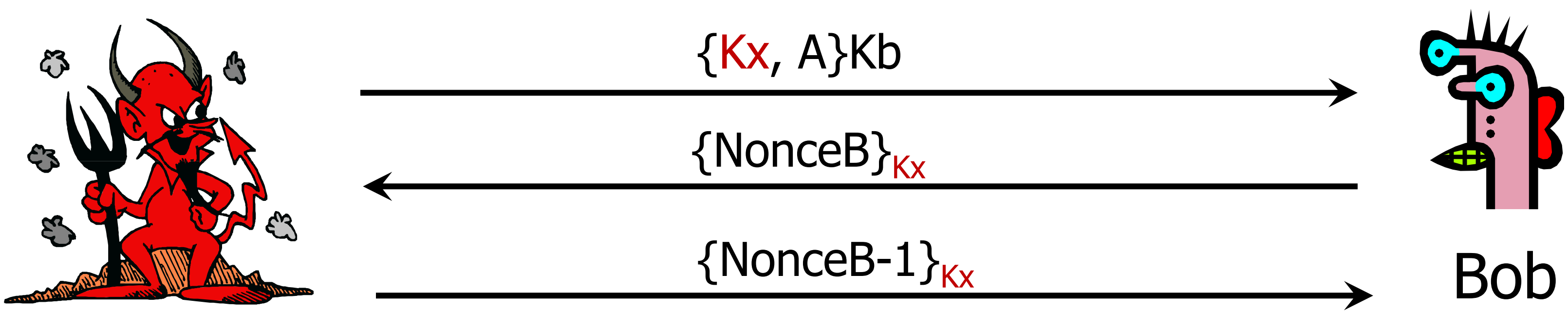
NS Symmetric-Key Protocol



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

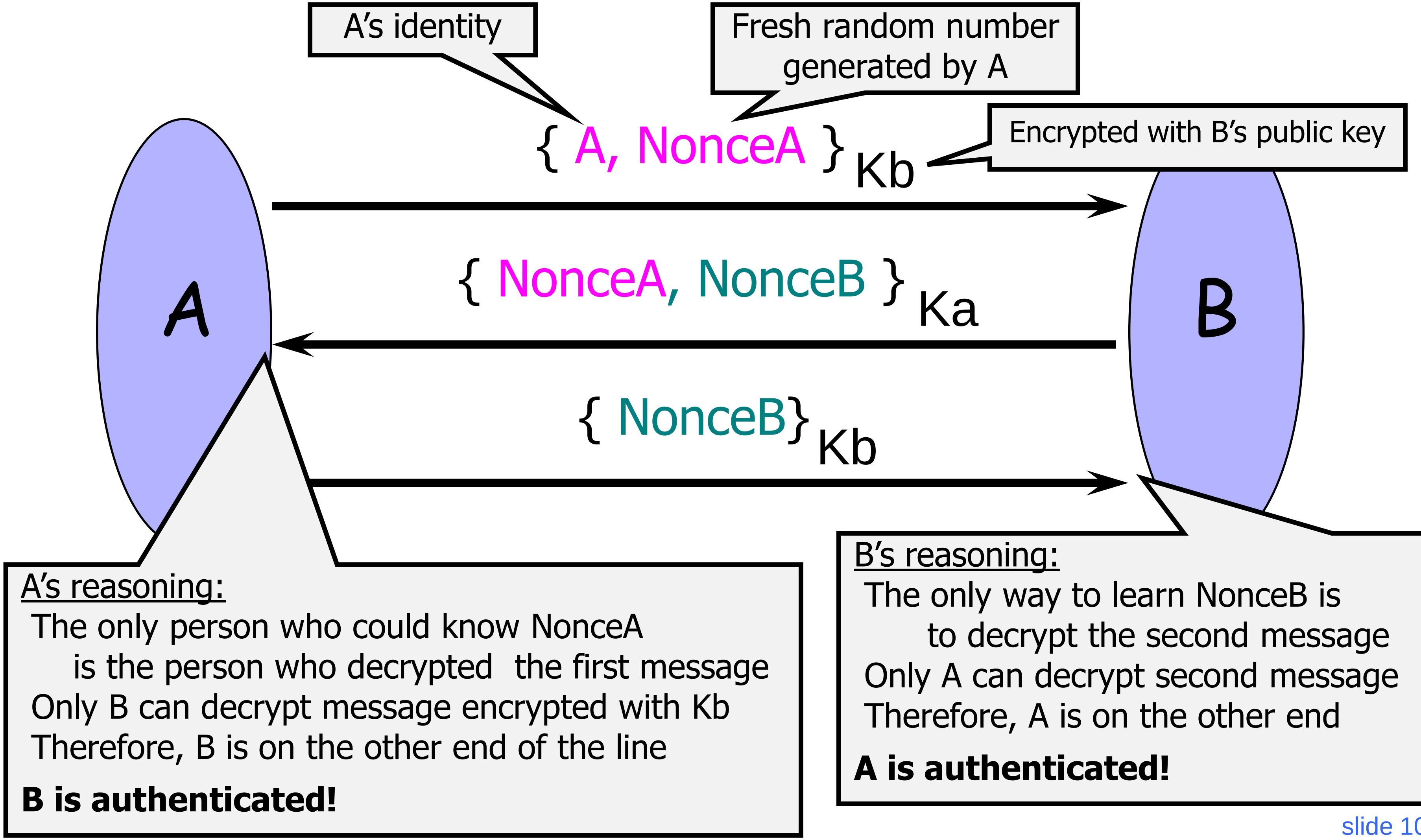
Denning-Sacco Attack

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

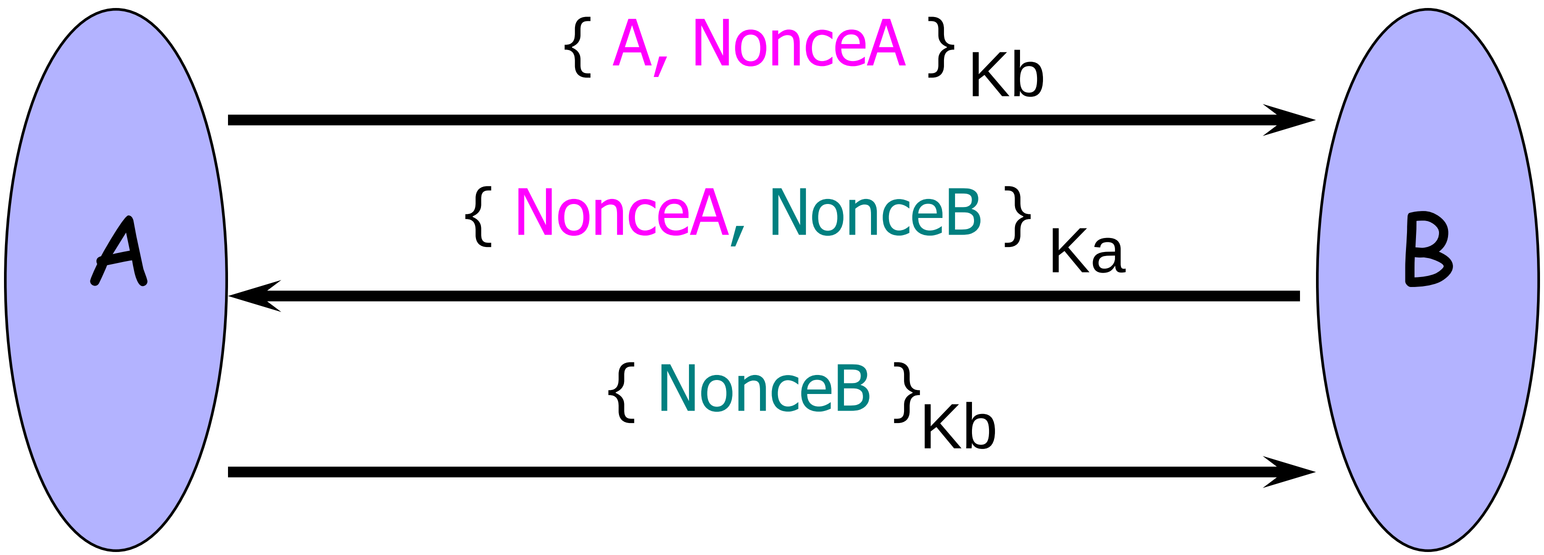


- B now believes he shares a fresh secret Kx with A
- Moral: use timestamps to detect replay of old messages

NS Public-Key Protocol



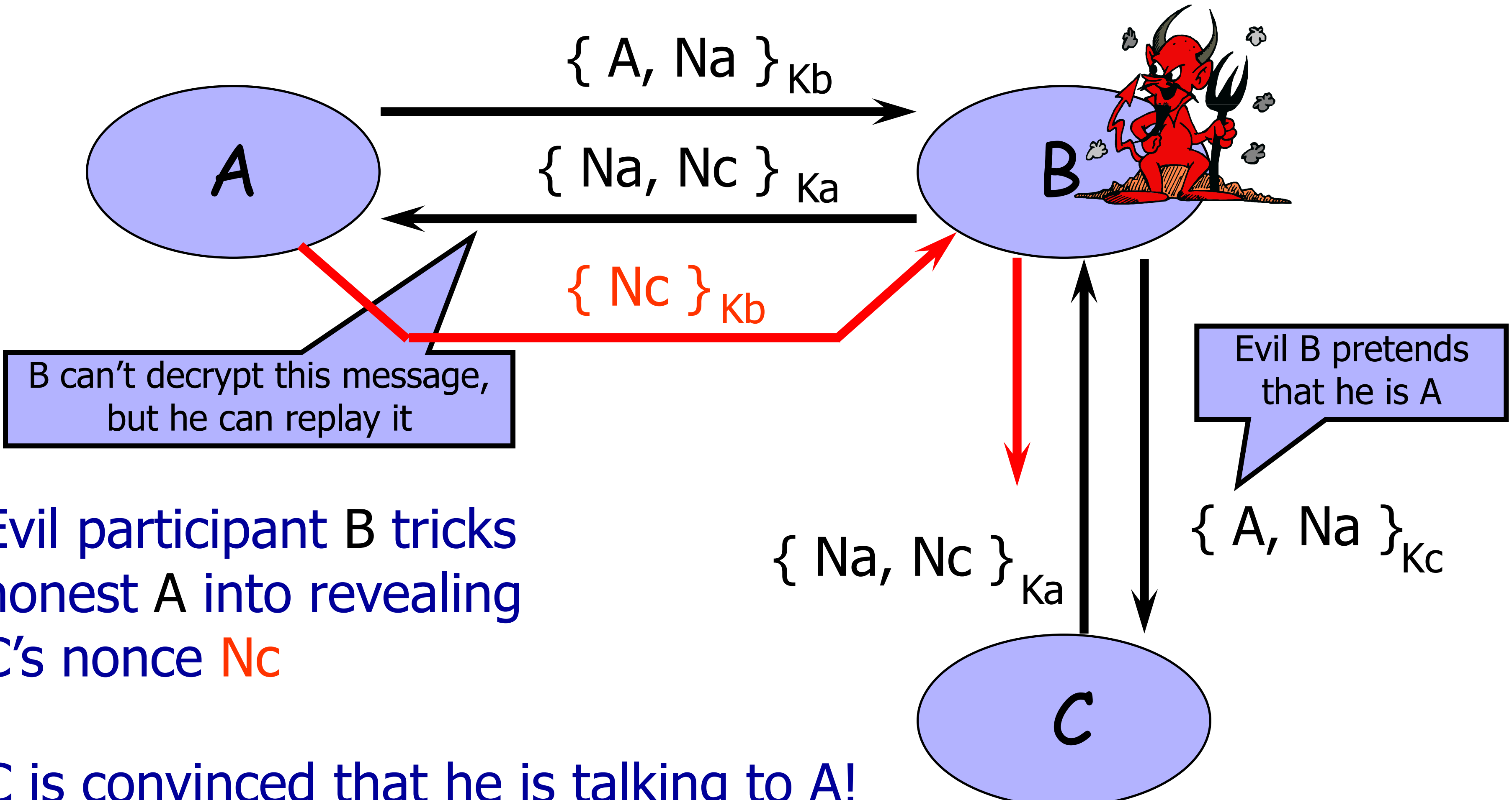
What Does This Protocol Achieve?



- 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

[Lowe. “Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR”. TACAS 1996]

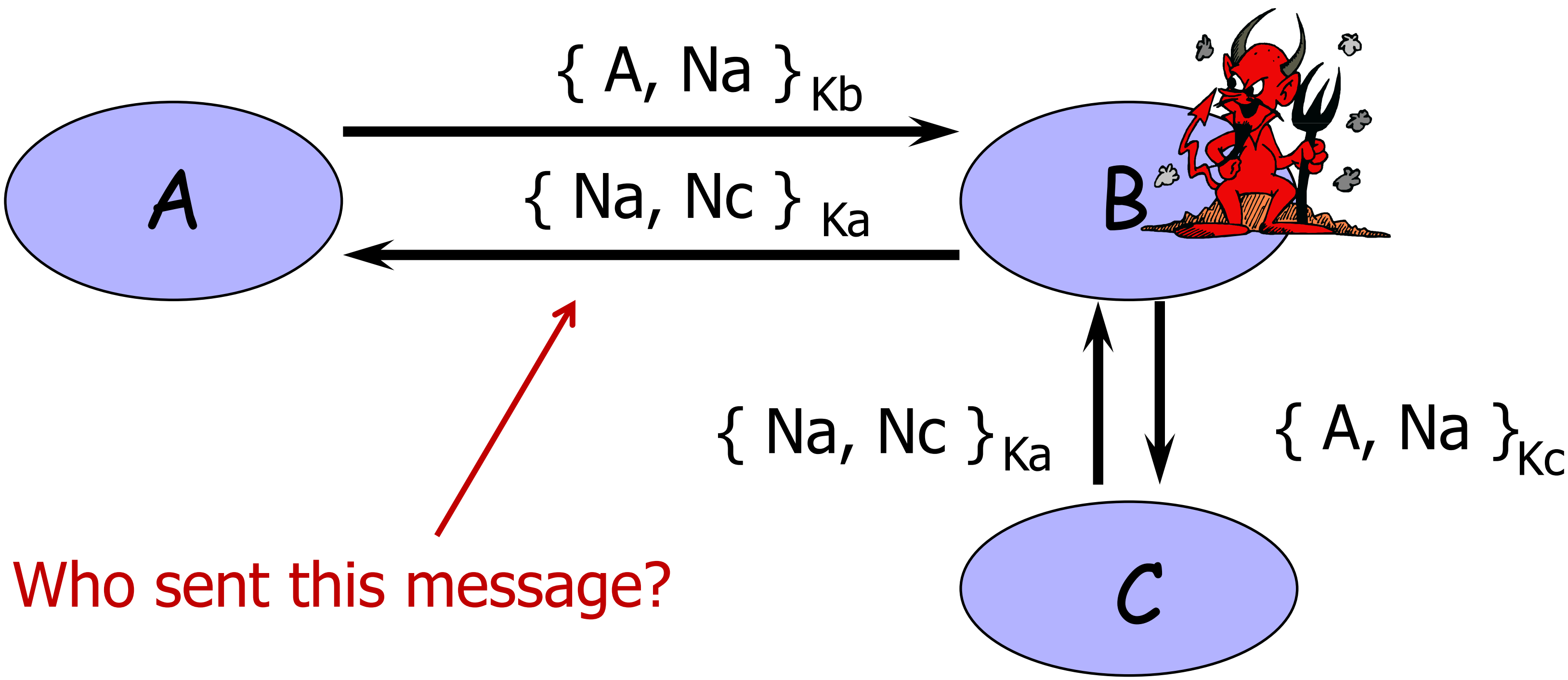


Evil participant B tricks honest A into revealing C’s nonce Nc

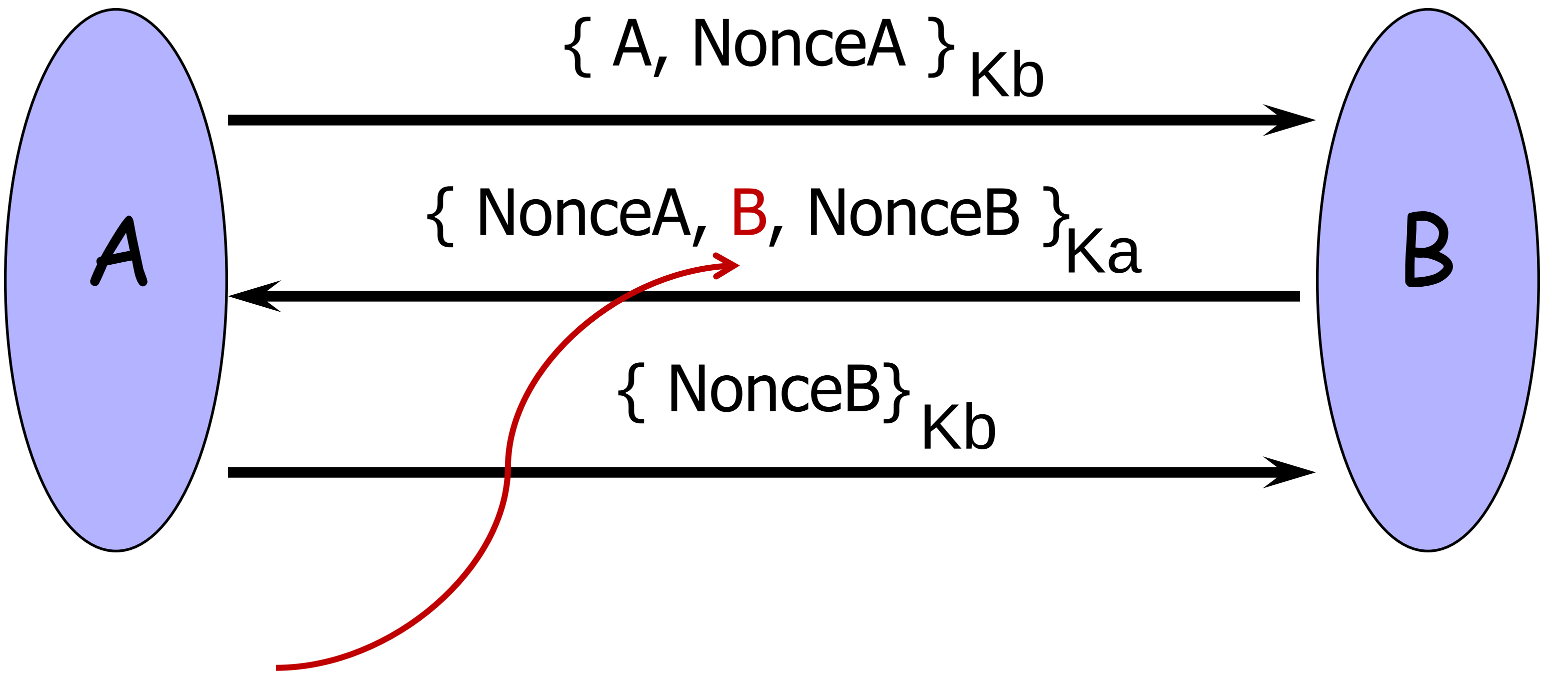
C is convinced that he is talking to A!

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

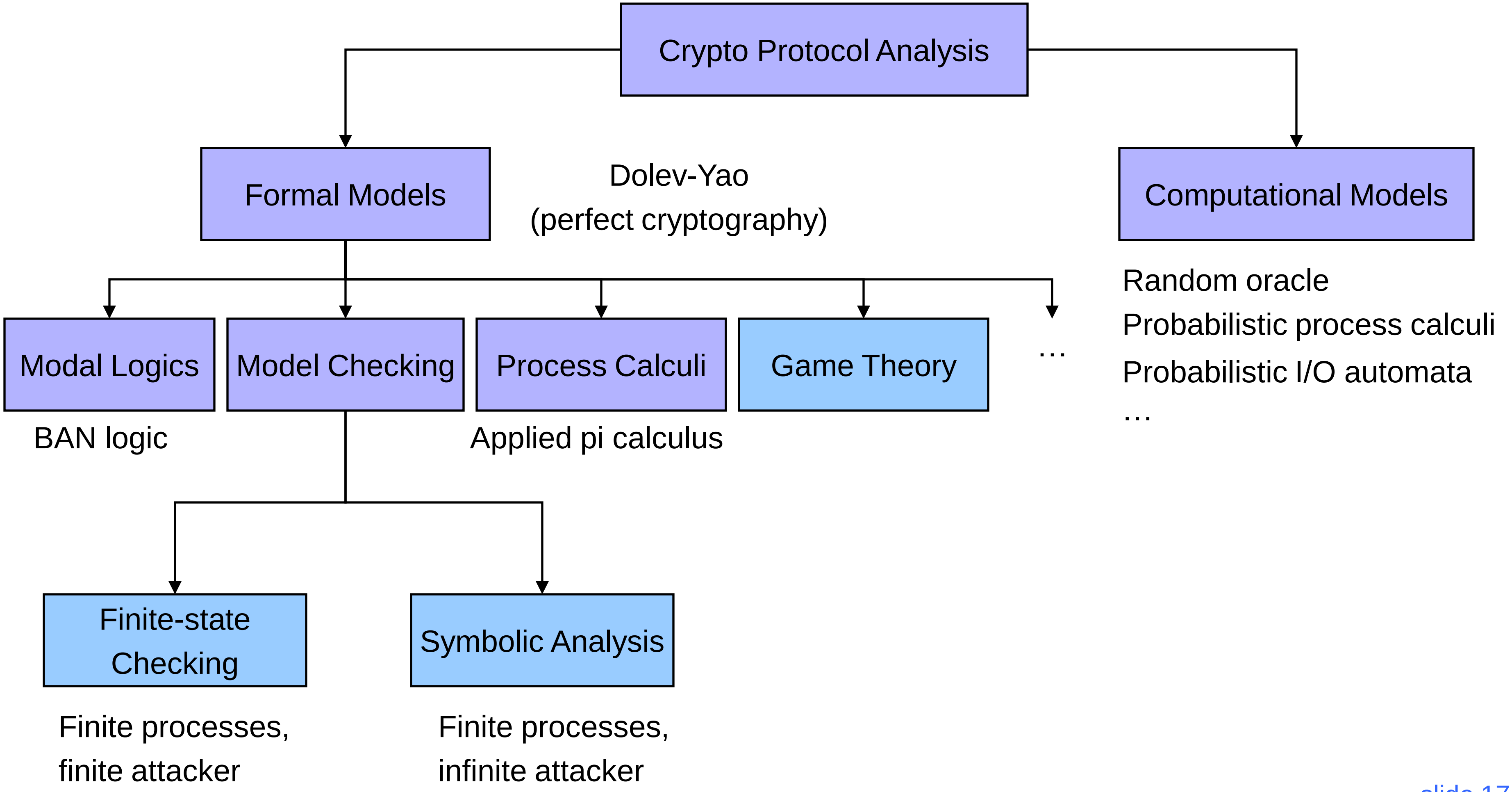
- Attacker is a legitimate protocol participant!
- Exploits participants' reasoning to fool them
  - A is correct that B must have decrypted $\{A,Na\}_{Kb}$ message, but this does not mean that the $\{Na,Nb\}_{Ka}$ message came from B
  - The attack does not rely on breaking cryptography!
- It is important 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

- Model protocol
- Model adversary
- Formally state security properties
- 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

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

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

```
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

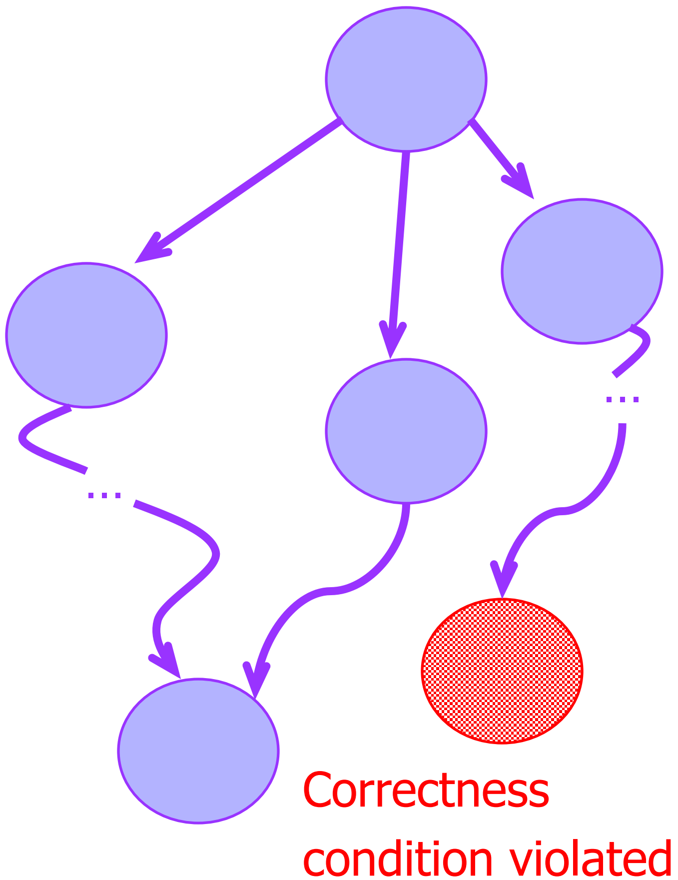
- Read and write on the network
  - Full control over all messages exchanged by honest parties (but cannot break cryptography)
- Analyze messages
  - Decrypt if and only if correct key is known
  - Break into smaller pieces
- Construct messages
  - Concatenate known fragments
  - Encrypt with known keys

Correctness Conditions

- Specified as predicates over system variables
- Secrecy
  - ! setInclusion(B.myNonce, Attacker.KnownNonces) &
  - ! setInclusion(A.myNonce, Attacker.KnownNonces)
- Authentication
  - ∀ 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

Stream Ciphers

- **One-time pad:**

  Ciphertext(Key,Message)=Message⊕Key

  - Key must be a random bit sequence as long as message
- Idea: replace "random" with "pseudo-random"
  - Use a pseudo-random number generator (PRNG)
  - PRNG takes a short, truly random secret seed and expands it into a long "random-looking" sequence
    - E.g., 128-bit seed into a 10^6-bit pseudo-random sequence

No efficient algorithm can tell this sequence from truly random

- Ciphertext(Key,Msg)=IV, Msg⊕PRNG(IV,Key)
  - Message processed bit by bit (unlike block cipher)

Stream Cipher Terminology

- The seed of a pseudo-random generator typically consists of initialization vector (IV) and key
  - The key is a secret known only to the sender and the recipient, not sent with the ciphertext
  - IV is usually sent with the ciphertext
- The pseudo-random bit stream produced by PRNG(IV,key) is referred to as the keystream
- Encrypt message by XORing with keystream
  - ciphertext = message $\oplus$ keystream

Properties of Stream Ciphers

- Usually very fast (faster than block ciphers)
  - Used where speed is important: WiFi, DVD, RFID, VoIP
- Unlike one-time pad, stream ciphers do not provide perfect secrecy
  - Only as secure as the underlying PRNG
  - If used properly, can be as secure as block ciphers
- PRNG must be cryptographically secure

Using Stream Ciphers

- No integrity
  - Associativity & commutativity: $(M_1 \oplus PRNG(seed)) \oplus M_2 = (M_1 \oplus M_2) \oplus PRNG(seed)$
  - Need an additional integrity protection mechanism
- Known-plaintext attack is very dangerous if keystream is ever repeated
  - Self-cancellation property of XOR: $X \oplus X = 0$
  - $(M_1 \oplus PRNG(seed)) \oplus (M_2 \oplus PRNG(seed)) = M_1 \oplus M_2$
  - If attacker knows M_1, then easily recovers M_2 … also, most plaintexts contain enough redundancy that can recover parts of both messages from $M_1 \oplus M_2$

How Random is "Random"?

Cryptographically Secure PRNG

- Next-bit test: given N bits of the pseudo-random sequence, predict $(N+1)^{st}$ bit
  - Probability of correct prediction should be very close to 1/2 for any efficient adversarial algorithm

    (means what?)
- PRNG state compromise
  - Even if the attacker learns the complete or partial state of the PRNG, he should not be able to reproduce the previously generated sequence
    - … or future sequence, if there'll be future random seed(s)
- Common PRNGs are not cryptographically secure

RC4

- Designed by Ron Rivest for RSA in 1987
- Simple, fast, widely used
  - SSL/TLS for Web security, WEP for wireless

```
Byte array S[256] contains a permutation of numbers from 0 to 255
i = j := 0
loop
    i := (i+1) mod 256
    j := (j+S[i]) mod 256
    swap(S[i],S[j])
    output (S[i]+S[j]) mod 256
end loop
```

RC4 Initialization

```
Divide key K into L bytes
for i = 0 to 255 do
    S[i] := i
j := 0
for i = 0 to 255 do
        j := (j+S[i]+K[i mod L]) mod 256
        swap(S[i],S[j])
```

Key can be any length up to 2048 bits

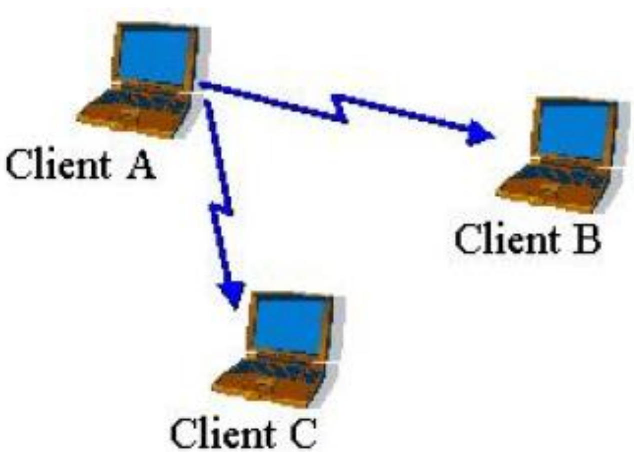
Generate initial permutation from key K

- To use RC4, usually prepend initialization vector (IV) to the key
  - IV can be random or a counter
- RC4 is not random enough… First byte of generated sequence depends only on 3 cells of state array S - this can be used to extract the key!
  - To use RC4 securely, RSA suggests discarding first 256 bytes

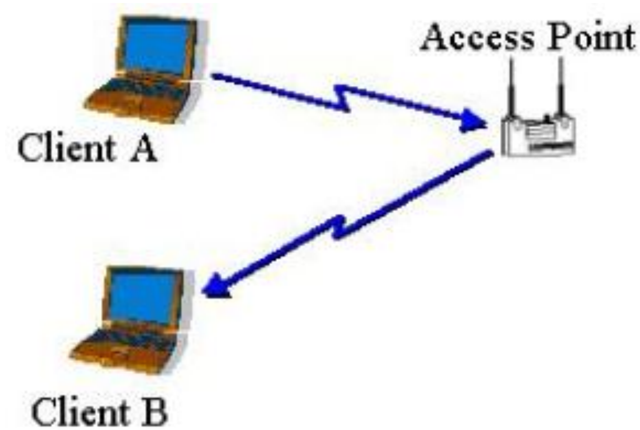
Fluhrer-Mantin-Shamir attack

802.11b Overview

- Standard for wireless networks (IEEE 1999)
- Two modes: infrastructure and ad hoc



IBSS (ad hoc) mode



BSS (infrastructure) mode

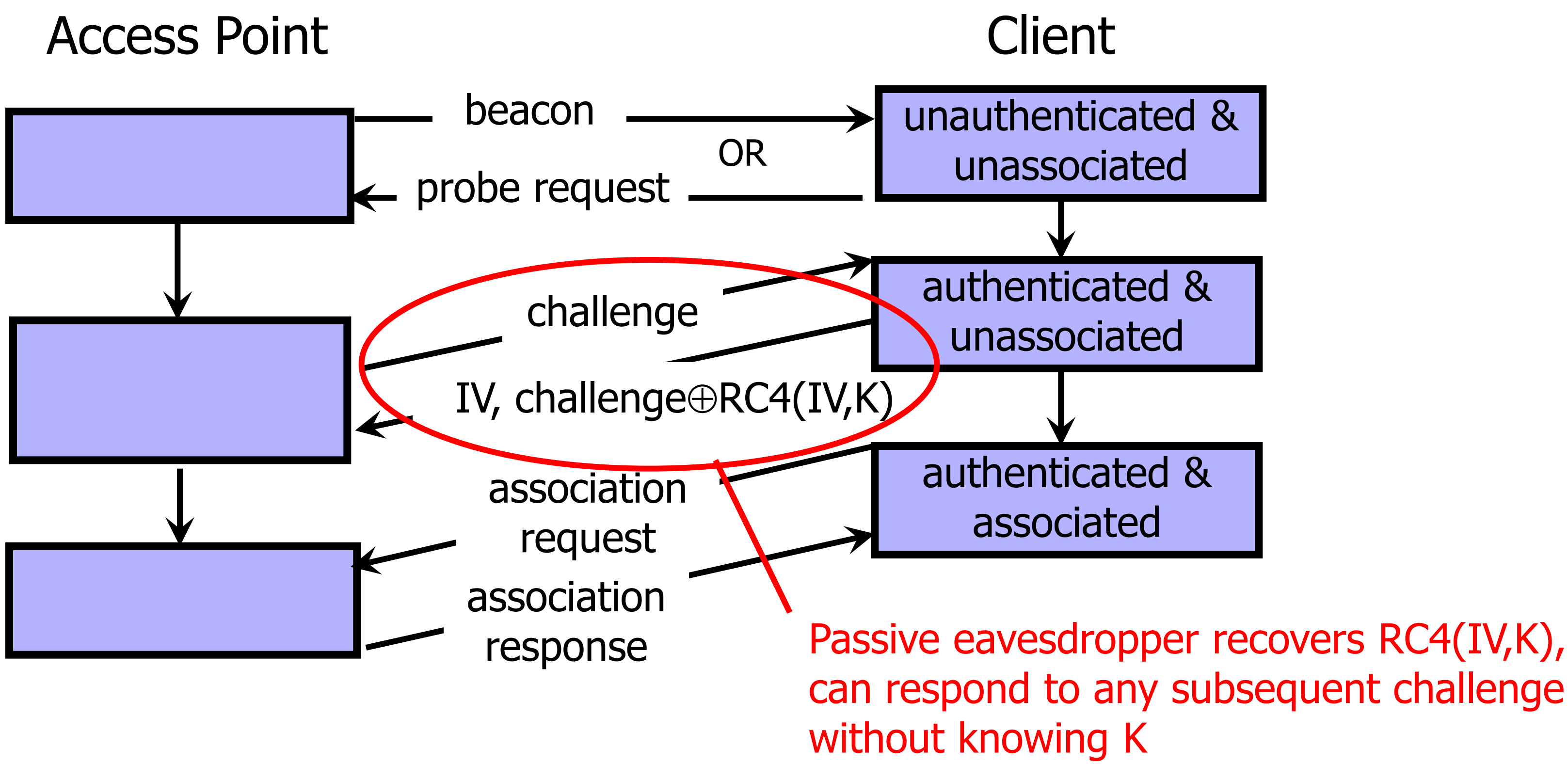
WEP: Wired Equivalent Privacy

- Special-purpose protocol for 802.11b
- Goals: confidentiality, integrity, authentication
  - Intended to make wireless as secure as wired network
- Assumes that a secret key is shared between access point and client
- Uses RC4 stream cipher seeded with 24-bit initialization vector and 40-bit key
  - Terrible design choice for wireless environment

Shared-Key Authentication

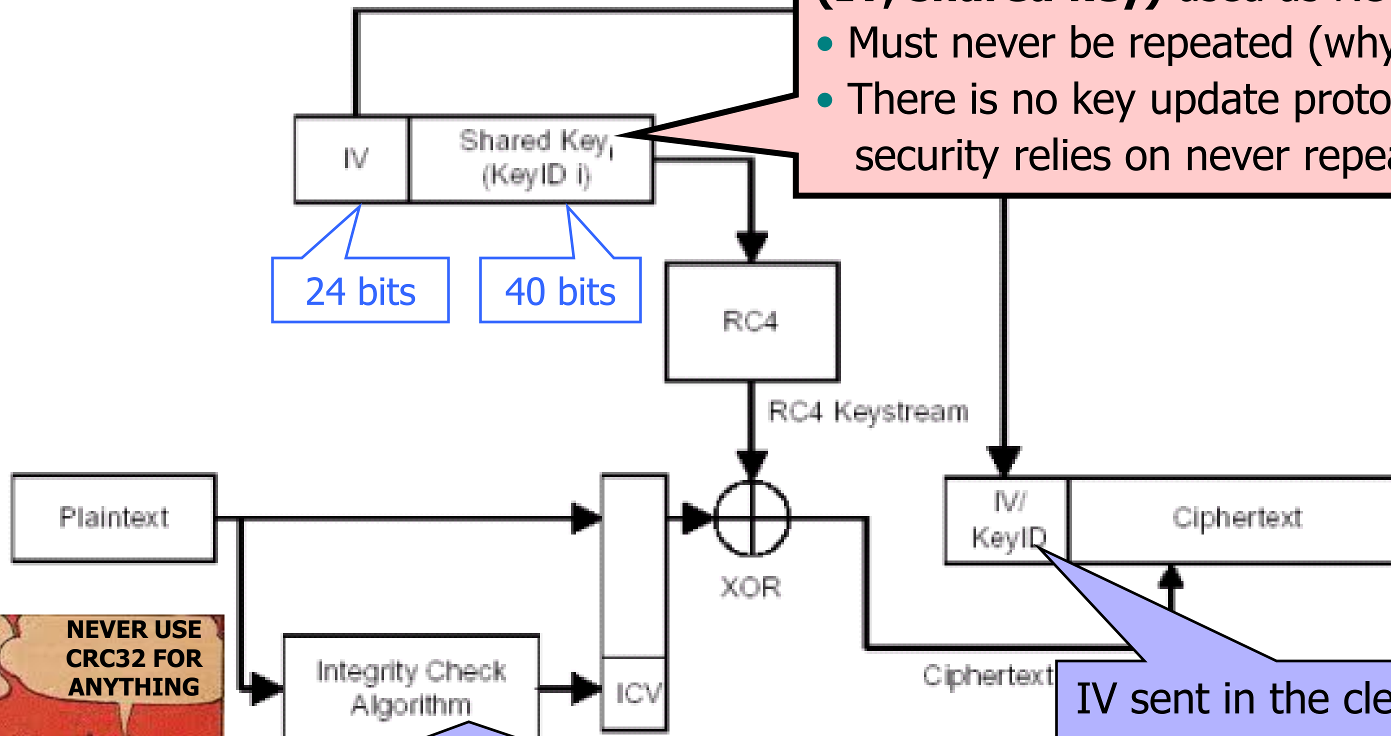
[Borisov et al. "Intercepting Mobile Communications: The Insecurity of 802.11". MOBICOM 2001]

Prior to communicating data, access point may require client to authenticate



How WEP Works

(IV, shared key) used as RC4 seed
- Must never be repeated (why?)
- There is no key update protocol, so security relies on never repeating IV



IV sent in the clear

Worse: <u>changing IV with each packet is optional!</u>

CRC-32 checksum is linear in ⊕:

if attacker flips some plaintext bits, he knows which bits of CRC to flip to produce the <u>same checksum</u>

no integrity!

Picture: iSEC Partners

RC4 Is a Bad Choice for Wireless

[Borisov et al.]

- Stream ciphers require sender and receiver to be at the same place in the keystream
  - Not suitable when packet losses are common
- WEP solution: a separate keystream for each packet (requires a separate seed for each packet)
  - Can decrypt a packet even if a previous packet was lost
- But there aren't enough possible seeds!
  - RC4 seed = 24-bit initialization vector + fixed key
  - Assuming 1500-byte packets at 11 Mbps, 2^{24} possible IVs will be exhausted in about 5 hours
- Seed reuse is deadly for stream ciphers

Recovering the Keystream

[Borisov et al.]

- Get access point to encrypt a known plaintext
  - Send spam, access point will encrypt and forward it
  - Get victim to send an email with known content
- With known plaintext, easy to recover keystream
  - $C \oplus M = (M \oplus RC4(IV,key)) \oplus M = RC4(IV,key)$
- Even without knowing the plaintext, can exploit plaintext regularities to recover partial keystream
  - Plaintexts are not random: for example, IP packet structure is very regular
- Not a problem if the keystream is not re-used

Keystream Will Be Re-Used

[Borisov et al.]

- In WEP, repeated IV means repeated keystream
- Busy network will repeat IVs often
  - Many cards reset IV to 0 when re-booted, then increment by 1 ⇒ expect re-use of low-value IVs
  - If IVs are chosen randomly, expect repetition in $O(2^{12})$ due to birthday paradox
- Recover keystream for each IV, store in a table
  - (KnownM ⊕ RC4(IV,key)) ⊕ KnownM = RC4(IV,key)
- Wait for IV to repeat, decrypt, enjoy plaintext
  - (M′ ⊕ RC4(IV,key)) ⊕ RC4(IV,key) = M′

It Gets Worse

- Misuse of RC4 in WEP is a design flaw with no fix
  - Longer keys do not help!
    - The problem is re-use of IVs, their size is fixed (24 bits)
  - Attacks are passive and very difficult to detect
- Perfect target for the Fluhrer et al. attack on RC4
  - Attack requires known IVs of a special form
  - WEP sends IVs in plaintext
  - Generating IVs as counters or random numbers will produce enough "special" IVs in a matter of hours
- This results in key recovery (not just keystream)
  - Can decrypt even ciphertexts whose IV is unique

Fixing the Problem

- Extensible Authentication Protocol (EAP)
  - Developers can choose their own authentication method
    - Passwords (Cisco EAP-LEAP), public-key certificates (Microsoft EAP-TLS), passwords OR certificates (PEAP), etc.
- 802.11i standard fixes 802.11b problems
  - Patch (TKIP): still RC4, but encrypts IVs and establishes new shared keys for every 10 KBytes transmitted
    - Use same network card, only upgrade firmware
    - Deprecated by the Wi-Fi alliance
  - Long-term: AES in CCMP mode, 128-bit keys, 48-bit IVs
    - Block cipher in a stream cipher-like mode