Most Common Cryptography Mistakes

3/8/2016


STOP
HAMMER
TIME.


You fell victim to one of the classic blunders!

#8: Key Re-use

- Don't use same key for both directions.
  - Risk: replay attacks

- Don't re-use same key for both encryption and authentication.

#7: Careful with Concatenation

- Common mistake: Hash(S||T)
  - "builtin" || "securely" = "built" || "insecurely"

Amazon Web Services

http://amazon.com/set?u=daw&n=David&t=U&m=…

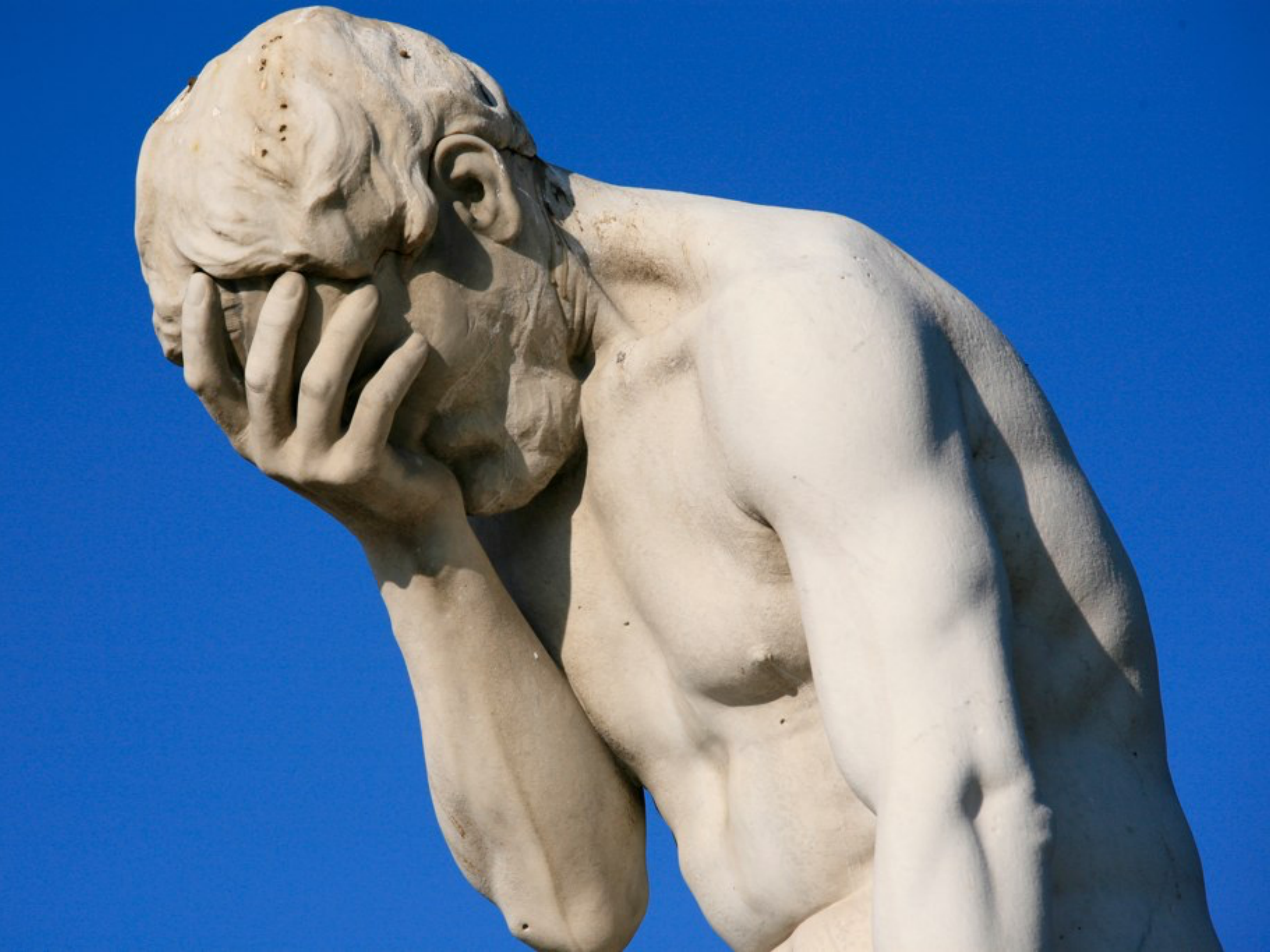
MAC(K,”udawnDavidtU”)

Amazon Web Services

://amazon.com/set?u=daw&n=DavidtAq&t=U&m=...

MAC(K,"udawnDavidtAqtU")

://amazon.com/set?u=daw&n=David&t=A&qt=U&m=...



#7: Careful with Concatenation

- Common mistake: Hash(S||T)
  - "builtin" || "securely" = "built" || "insecurely"
- Fix: Hash(len(S) || S || T)
- Make sure inputs to hash/MAC are uniquely decodable

#5: Don't Encrypt without Auth

- Common mistake: encrypt, but no authentication
  - A checksum does not provide authentication
- If you're encrypting, you probably want authenticated encryption
  - Encrypt-then-authenticate: $E_{k1}(M)$, $F_{k2}(E_{k1}(M))$
  - Or, use a dedicated AE mode: GCM, EAX, …

Encrypt without Auth Hall of Shame

- ASP.NET (x2)
- XML encryption
- Amazon EC2
- JavaServer Faces
- Ruby on Rails
- OWASP ESAPI
- IPSEC
- WEP
- SSH2

#4: Be Careful with Randomness

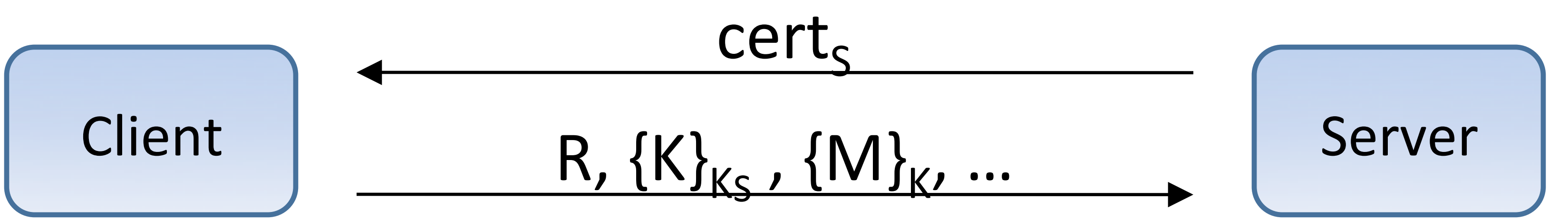
- Common mistake: use predictable random number generator (e.g., to generate keys)
- Solution: Use a crypto-quality PRNG.
  - /dev/urandom, CryptGenRandom, …

Netscape Navigator

```
char chall[16], k[16];

srand(getpid() + time(NULL)
      + getppid());
for (int i=0; i<16; i++)
  chal[i] = rand();
for (int i=0; i<16; i++)
  chal[i] = rand();
```

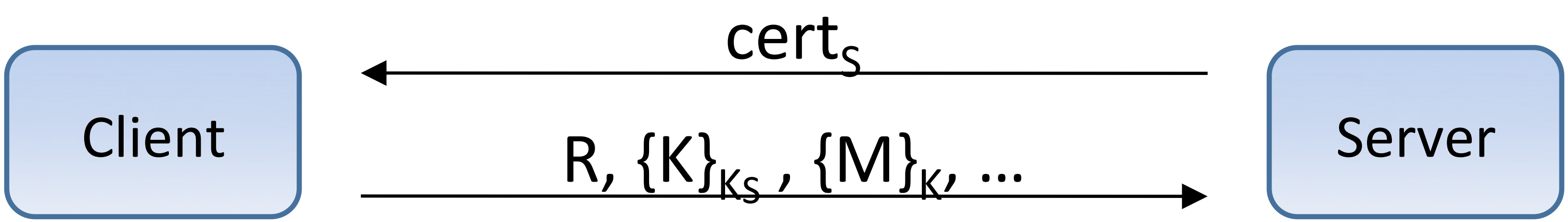
Netscape Navigator 1.1



where $(R, K) = hash(microseconds, x)$

$x = seconds + pid + (ppid << 12)$

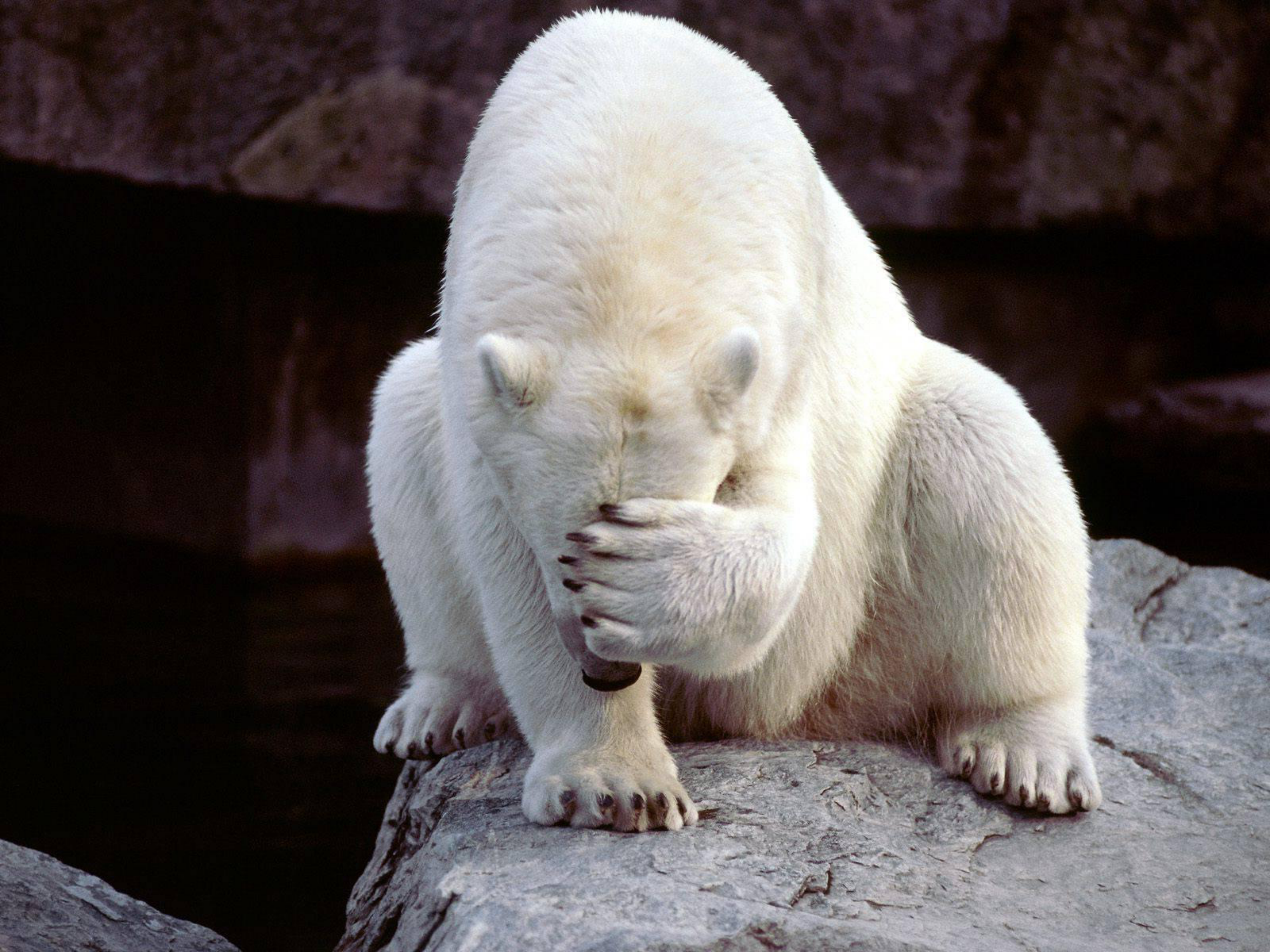
Netscape Navigator 1.1



where (R, K) = hash(microseconds, x)

x = seconds + pid + (ppid << 12)

Attack: Eavesdropper can guess x (≈ 10 bits) and microseconds (20 bits), and use R to check guess.



Bad PRNGs = broken crypto

- Netscape server’s private keys (≈ 32 bits)
- Kerberos v4’s session keys (≈ 20 bits)
- X11 MIT-MAGIC-COOKIE1 (8 bits)
- Linux vtun (≈ 1 bit)
- PlanetPoker site (≈ 18 bits)
- Debian OpenSSL (15 bits)
- CryptoAG – NSA spiked their PRNG
- Dual_EC_DRBG – backdoor that only NSA can use

#3: Passphrases Make Poor Keys

- Common mistake: Generate crypto key as Hash(passphrase)
- Problem: ≈ 20 bits of entropy; even with a slow hash, this is not nearly enough. Human-generated secrets just don't have enough entropy.
- Example: Bitcoin brainwallets
- Solution: Crypto keys should be random.

#2: Be Secure By Default

- Common mistake: Security is optional, or configurable, or negotiable
- Fix: There is one mode of operation, and it is secure. No human configuration needed.
  - e.g., Skype

Wardriving / Access Point Mapping

468 WEP

1,265 Clear

1,733 Total

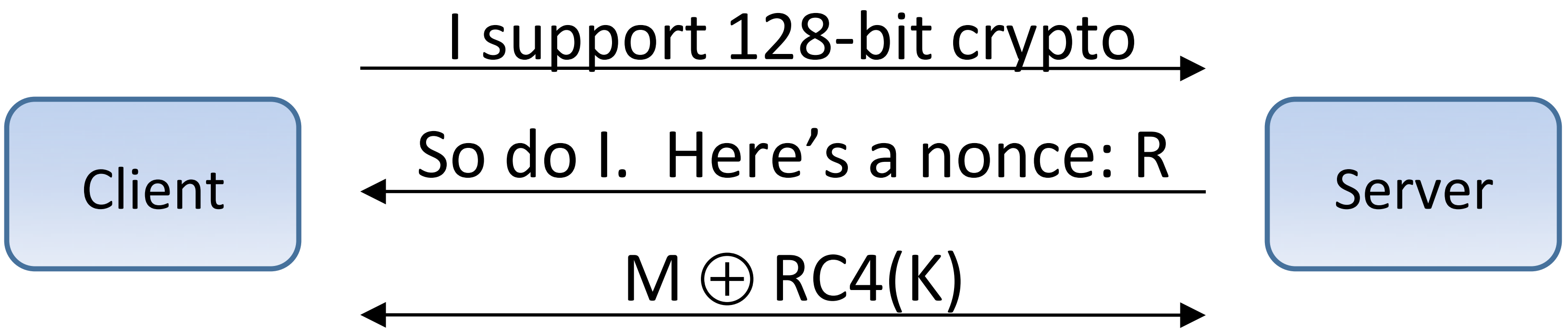
#2: Beware Rollback Attacks

- Common mistake: Security is negotiable, and attacker can persuade you to fall back to insecure crypto

A CASE STUDY

MS Point-to-Point Encryption (MPPE)

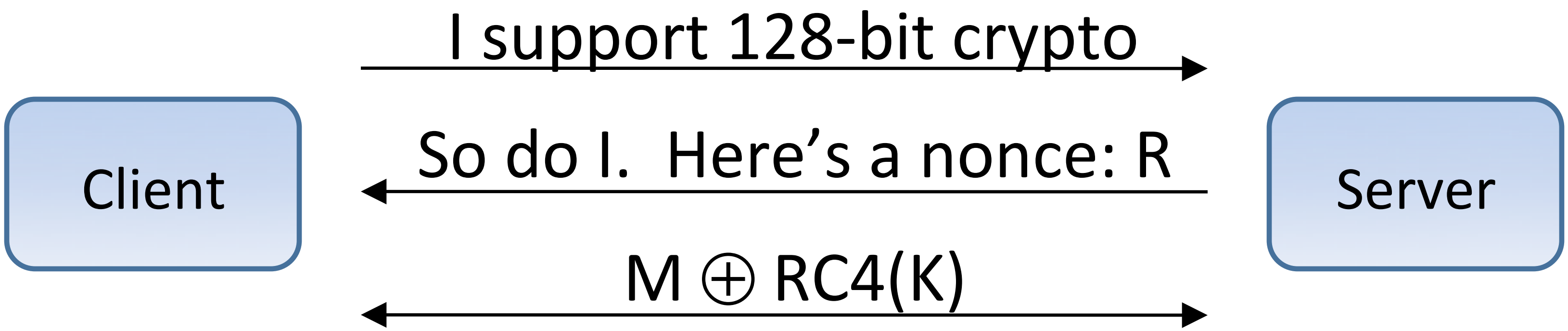
If both endpoints support 128-bit crypto:



where K = hash(password || R)

MS Point-to-Point Encryption (MPPE)

If both endpoints support 128-bit crypto:

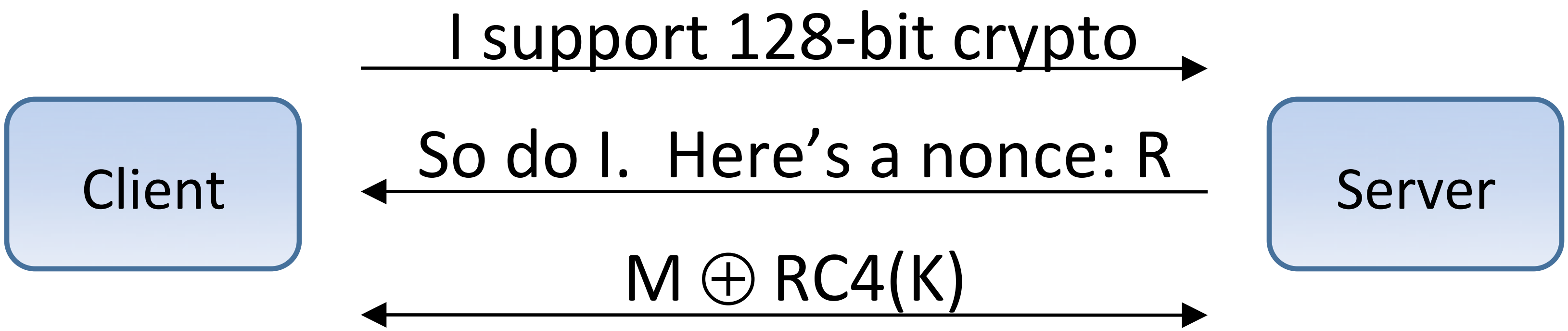


where K = hash(password || R)

Attack 1: Eavesdropper can try dictionary search on password, given some known plaintext.

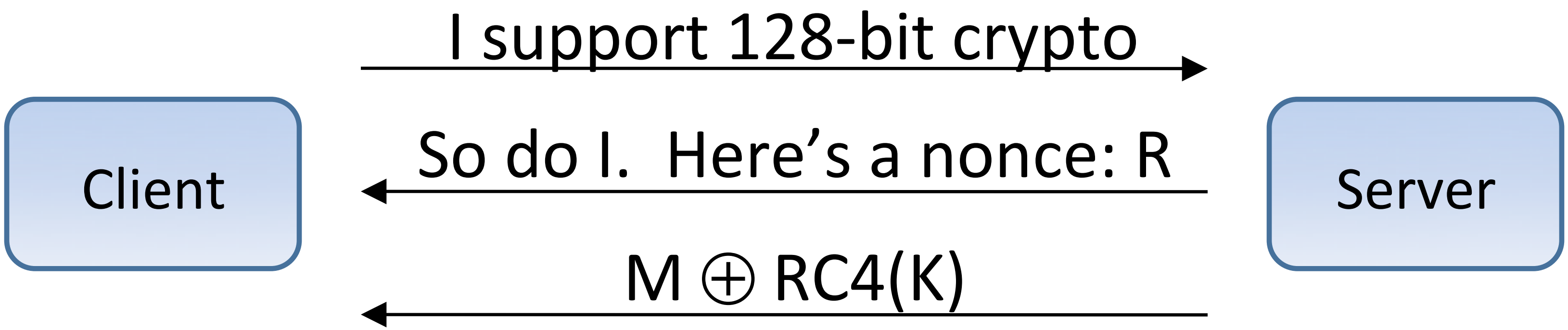
MS Point-to-Point Encryption (MPPE)

If both endpoints support 128-bit crypto:

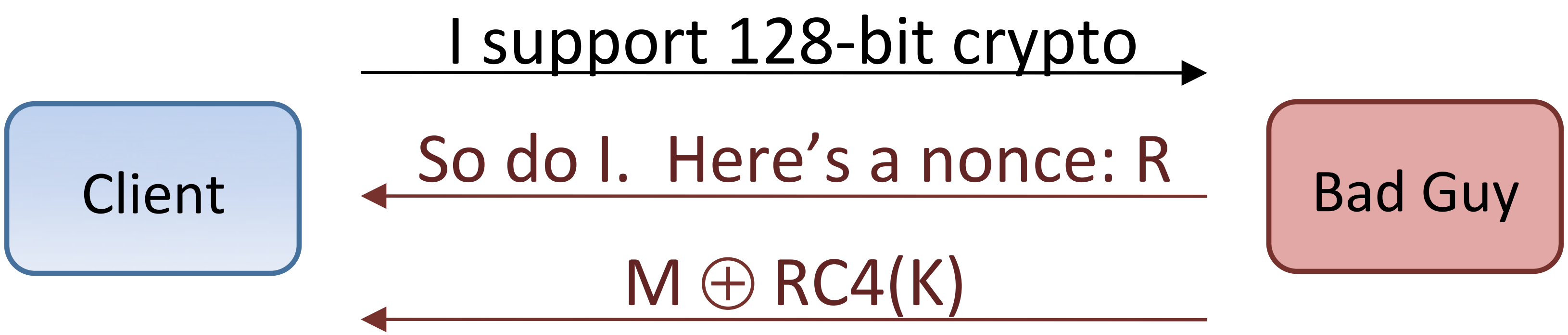


where K = hash(password || R)

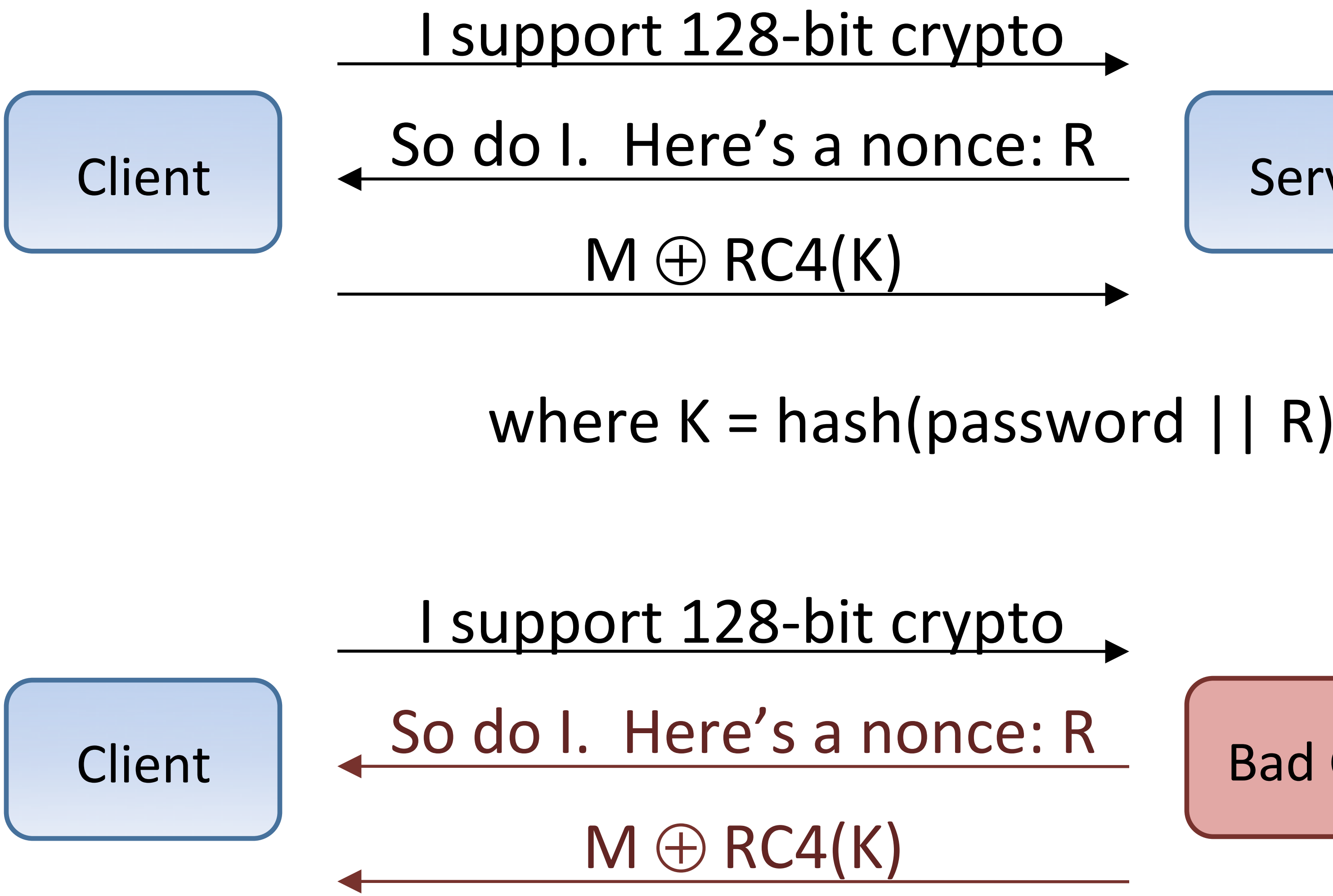
Attack 2: Active attacker can tamper with packets by flipping bits, since there is no MAC.



where K = hash(password || R)



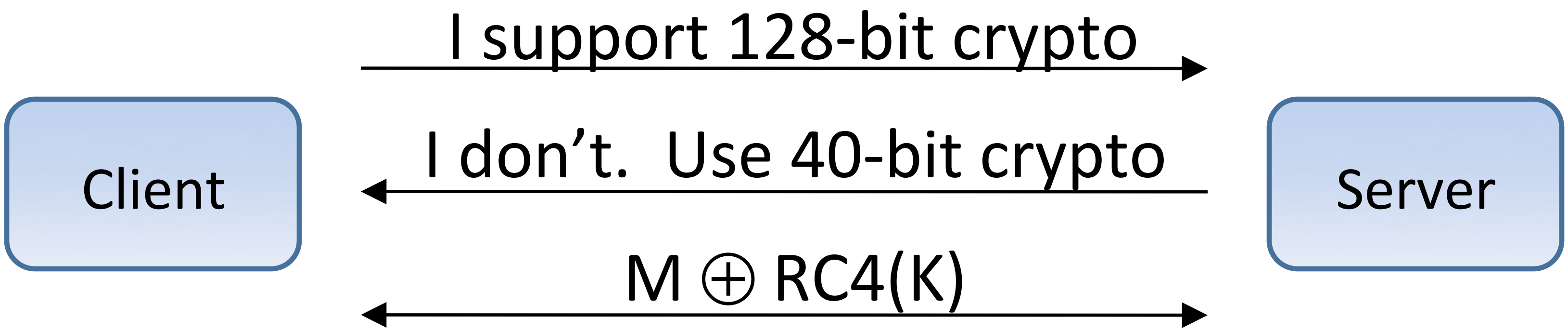
Attack 3: Bad guy can replay a prior session, since client doesn't contribute a nonce.



Attack 4: Bad guy can replay and reverse message direction, since same key used in both directions.

MS Point-to-Point Encryption (MPPE)

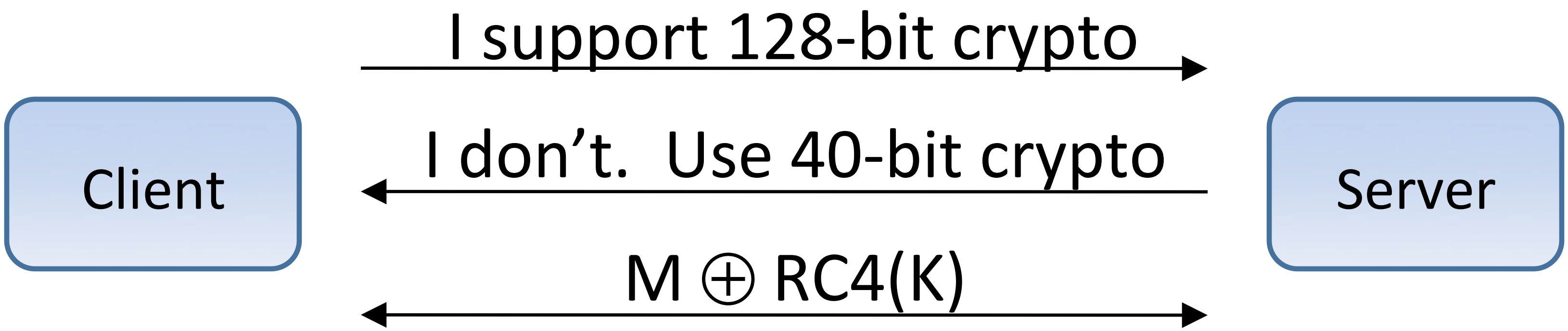
If one endpoint doesn't support 128-bit crypto:



where K = hash(uppercase(password))

MS Point-to-Point Encryption (MPPE)

If one endpoint doesn't support 128-bit crypto:

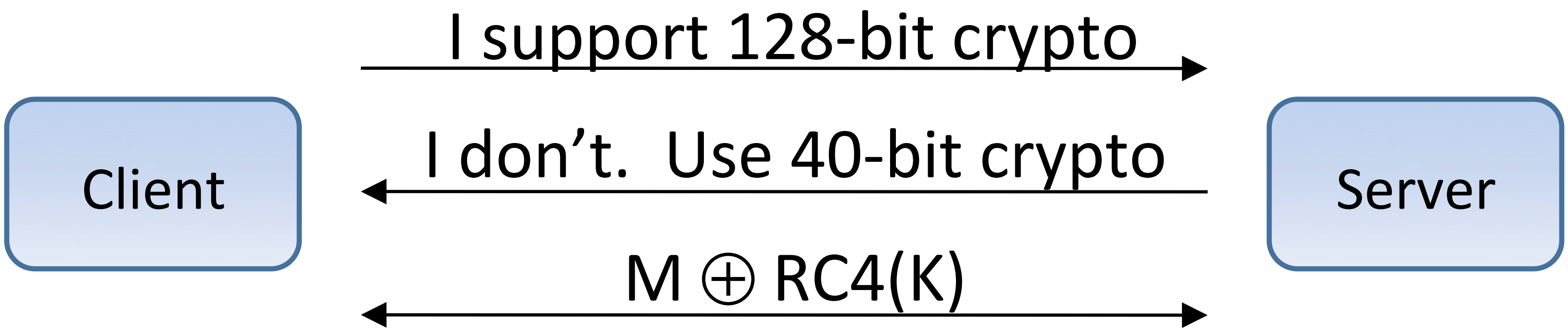


where K = hash(uppercase(password))

Attack 1: Eavesdropper can try dictionary search on password, given some known plaintext.

MS Point-to-Point Encryption (MPPE)

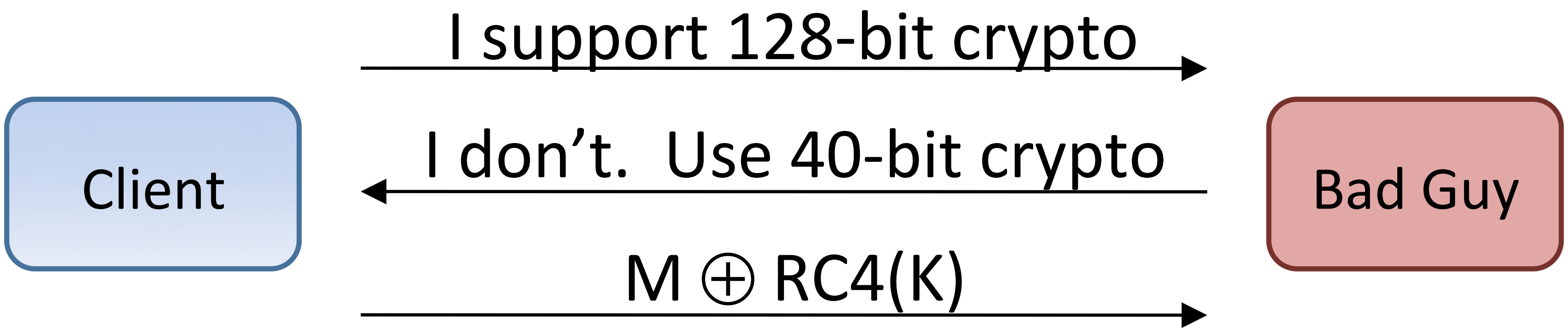
If one endpoint doesn't support 128-bit crypto:



where K = hash(uppercase(password))

Attack 2: Dictionary search can be sped up with precomputed table (given known plaintext).

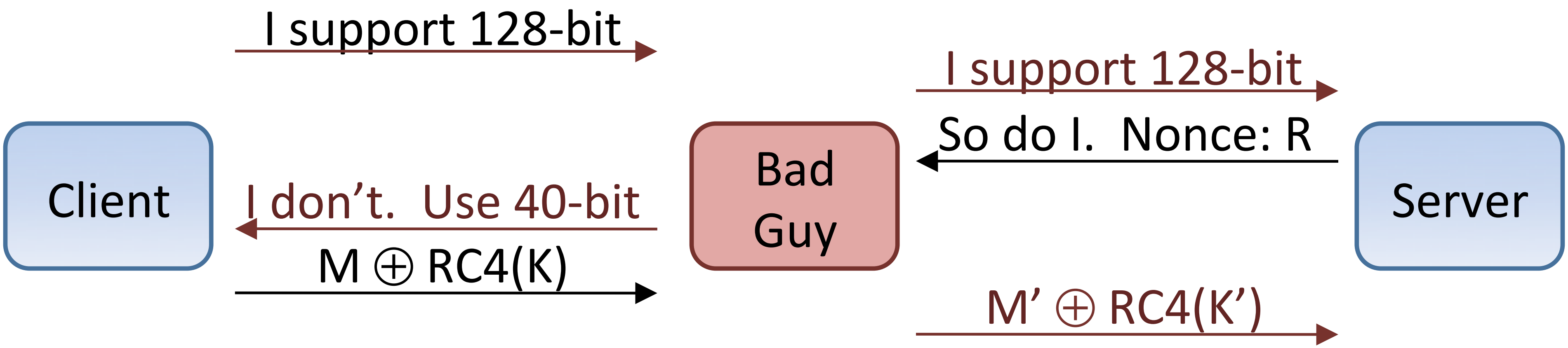
MS Point-to-Point Encryption (MPPE)



where K = hash(uppercase(password))

Attack 3: Imposter server can downgrade client to 40-bit crypto, then crack password.

MS Point-to-Point Encryption (MPPE)



where K = hash(uppercase(password)),
K' = hash(password || R)

Attack 4: Man-in-the-middle can downgrade crypto strength even if both client + server support 128-bit crypto, then crack password.

#1: Don’t Roll Your Own

- Don’t design your own crypto algorithm
- Use a time-honored, well-tested system
  - For data in transit: TLS, SSH, IPSEC
  - For data at rest: GnuPG

#0: Crypto Ain't Magic

"If you think cryptography is the solution to your problem, then you don't understand cryptography and you don't understand your problem."

– Roger Needham

Meta-Lessons

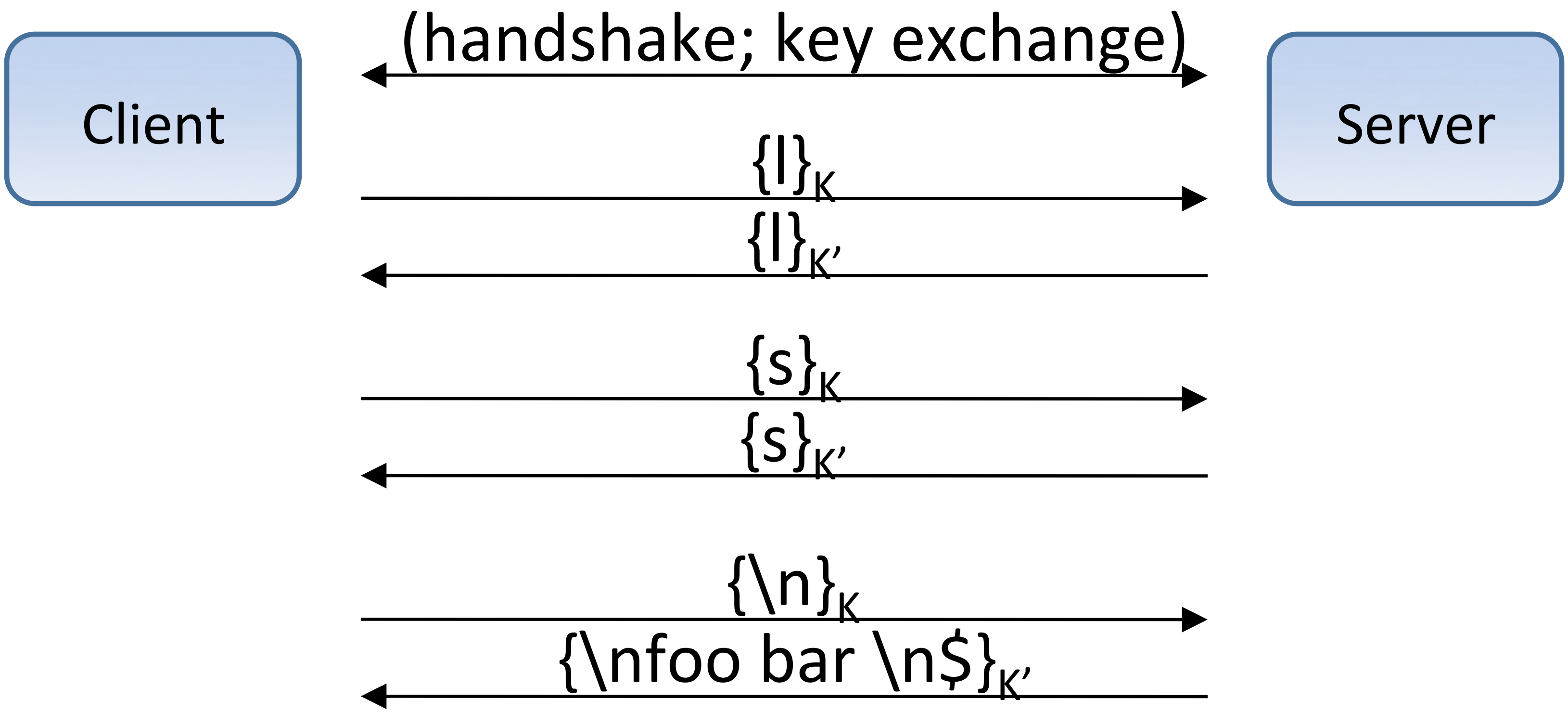
- Cryptography is hard.
- Hire an expert, or use an existing system (e.g., SSL, SSH, GnuPG).
- But: Most vulnerabilities are in applications and software, not in crypto algorithms.

BONUS MATERIAL

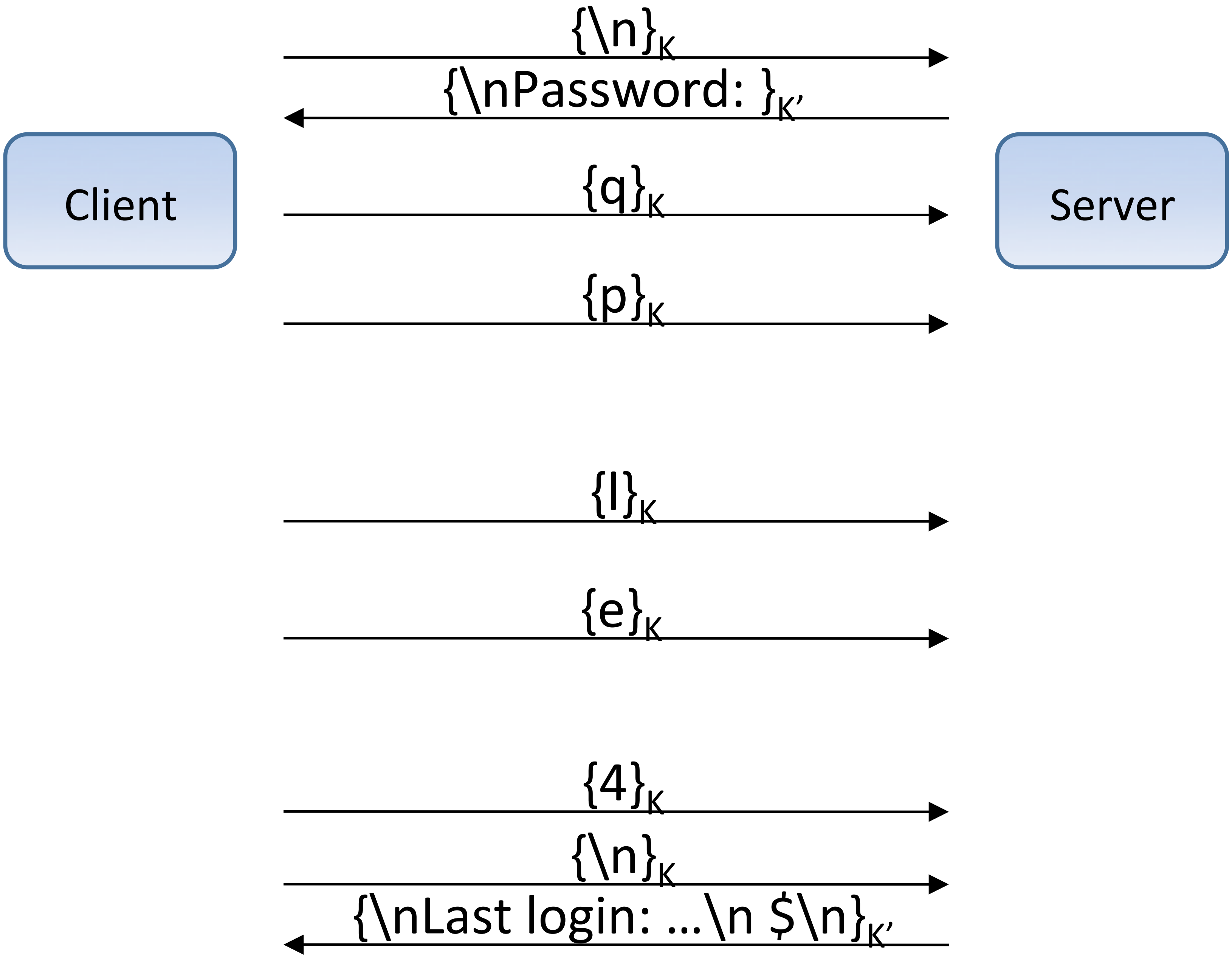
#8: Traffic Analysis is Still Possible

- Encryption doesn't hide sender, recipient, length, or time of message. ("meta-data")

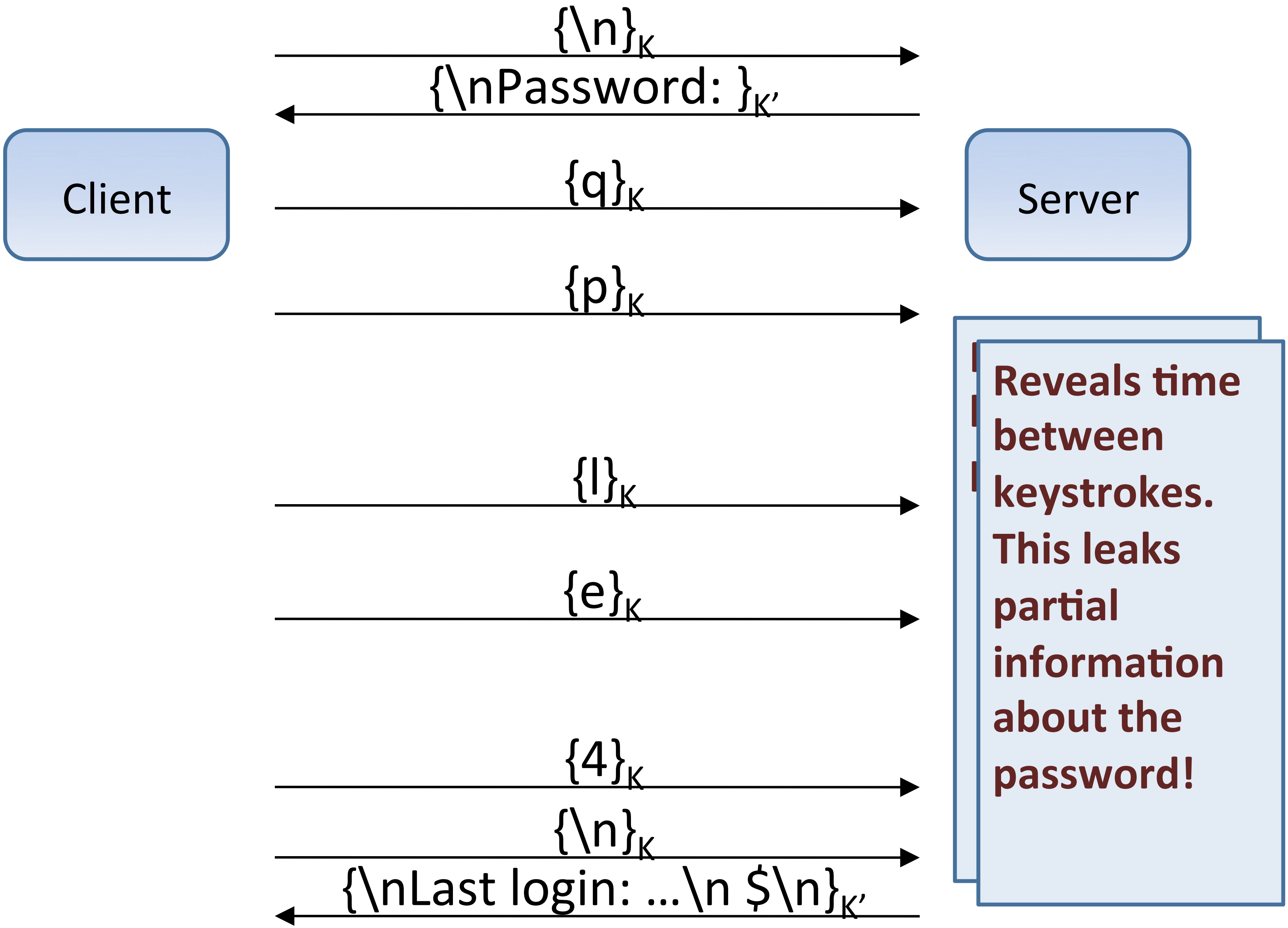
SSH


Client
Server
(handshake; key exchange)
{l}K
{l}K’
{s}K
{s}K’
{\n}K
{\nfoo bar \n$}K’

SSH



SSH


{\n}K
{\nPassword: }K'
Client
Server
{q}K
{p}K
{l}K
{e}K
{4}K
{\n}K
{\nLast login: …\n $\n}K'
Reveals time between keystrokes. This leaks partial information about the password!

Lessons Summarized

- Don’t design your own crypto algorithm.
- Use authenticated encryption (don’t encrypt without authenticating).
- Use crypto-quality random numbers.
- Don’t derive crypto keys from passphrases.
- Be secure by default.
- Be careful with concatenation.
- Don’t re-use nonces/IVs. Don’t re-use keys for multiple purposes.
- Encryption doesn’t prevent traffic analysis (“metadata”).

#7: Don’t re-use nonces/IVs

- Re-using a nonce or IV leads to catastrophic security failure.

Credit card numbers in a database

dgaTkyuPS8bs4rPXoQn3
dgaalSeET8Hv4rvfpQrz
cQGakyuFQcri6brfoAH6Jg==
dgWdmSuESsro4bfXpQj0
cQSYmCKLScDt4bDXqAj2Ig==
cQWTlCKNSsfr5bDfqAnzIw==
cAKdkyOMT8Ti6LvQpwj2IA==

After Base64 decoding

76	06	93	93	2b	8f	4b	c6	ec	e2	b3	d7	a1	09	f7	
76	06	9a	95	27	84	4f	c1	ef	e2	bb	df	a5	0a	f3	
71	01	9a	93	2b	85	41	ca	e2	e9	ba	df	a0	01	fa	26
76	05	9d	99	2b	84	4a	ca	e8	e1	b7	d7	a5	08	f4	
71	04	98	98	22	8b	49	c0	ed	e1	b0	d7	a8	08	f6	22
71	05	93	94	22	8d	4a	c7	eb	e5	b0	df	a8	09	f3	23
70	02	9d	93	23	8c	4f	c4	e2	e8	bb	d0	a7	08	f6	20

Encrypted credit card numbers

76	06	93	93	2b	8f	4b	c6	ec	e2	b3	d7	a1	09	f7	
76	06	9a	95	27	84	4f	c1	ef	e2	bb	df	a5	0a	f3	
71	01	9a	93	2b	85	41	ca	e2	e9	ba	df	a0	01	fa	26
76	05	9d	99	2b	84	4a	ca	e8	e1	b7	d7	a5	08	f4	
71	04	98	98	22	8b	49	c0	ed	e1	b0	d7	a8	08	f6	22
71	05	93	94	22	8d	4a	c7	eb	e5	b0	df	a8	09	f3	23
70	02	9d	93	23	8c	4f	c4	e2	e8	bb	d0	a7	08	f6	20

Encrypted credit card numbers

76	06	93	93	2b	8f	4b	c6	ec	e2	b3	d7	a1	09	f7	
76	06	9a	95	27	84	4f	c1	ef	e2	bb	df	a5	0a	f3	
71	01	9a	93	2b	85	41	ca	e2	e9	ba	df	a0	01	fa	26
76	05	9d	99	2b	84	4a	ca	e8	e1	b7	d7	a5	08	f4	
71	04	98	98	22	8b	49	c0	ed	e1	b0	d7	a8	08	f6	22
71	05	93	94	22	8d	4a	c7	eb	e5	b0	df	a8	09	f3	23
70	02	9d	93	23	8c	4f	c4	e2	e8	bb	d0	a7	08	f6	20

ASCII: …, ‘3’ = 0x33, ‘4’ = 0x34, ‘5’ = 0x35, …

Encrypted credit card numbers

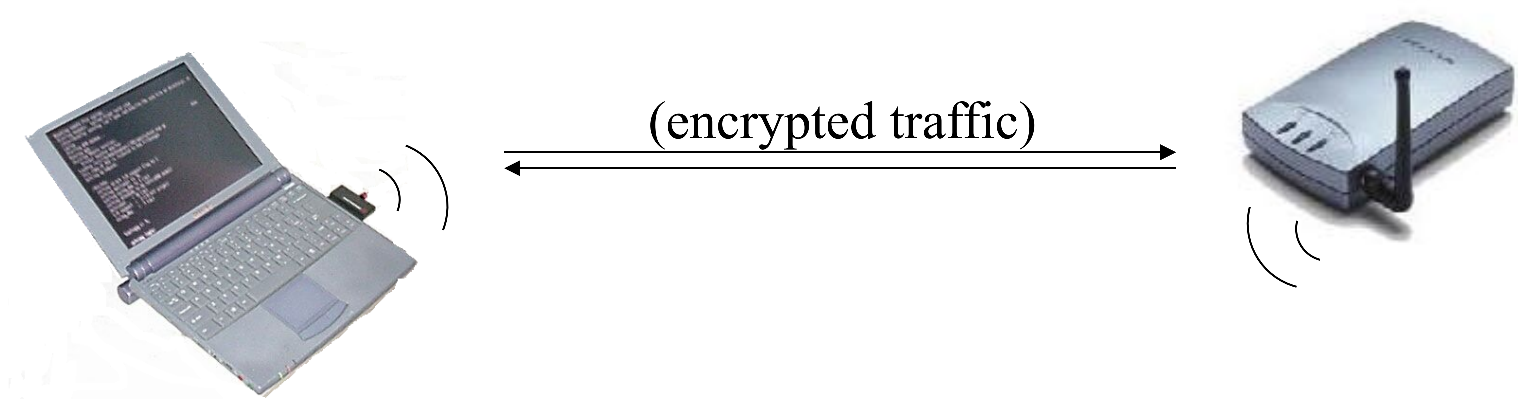
76	06	93	93	2b	8f	4b	c6	ec	e2	b3	d7	a1	09	f7	
76	06	9a	95	27	84	4f	c1	ef	e2	bb	df	a5	0a	f3	
71	01	9a	93	2b	85	41	ca	e2	e9	ba	df	a0	01	fa	26
76	05	9d	99	2b	84	4a	ca	e8	e1	b7	d7	a5	08	f4	
71	04	98	98	22	8b	49	c0	ed	e1	b0	d7	a8	08	f6	22
71	05	93	94	22	8d	4a	c7	eb	e5	b0	df	a8	09	f3	23
70	02	9d	93	23	8c	4f	c4	e2	e8	bb	d0	a7	08	f6	20

ASCII: ‘0’ = 0x30, …, ‘7’ = 0x37, ‘8’ = 0x38, ‘9’ = 0x39

#7: Don’t re-use nonces/IVs

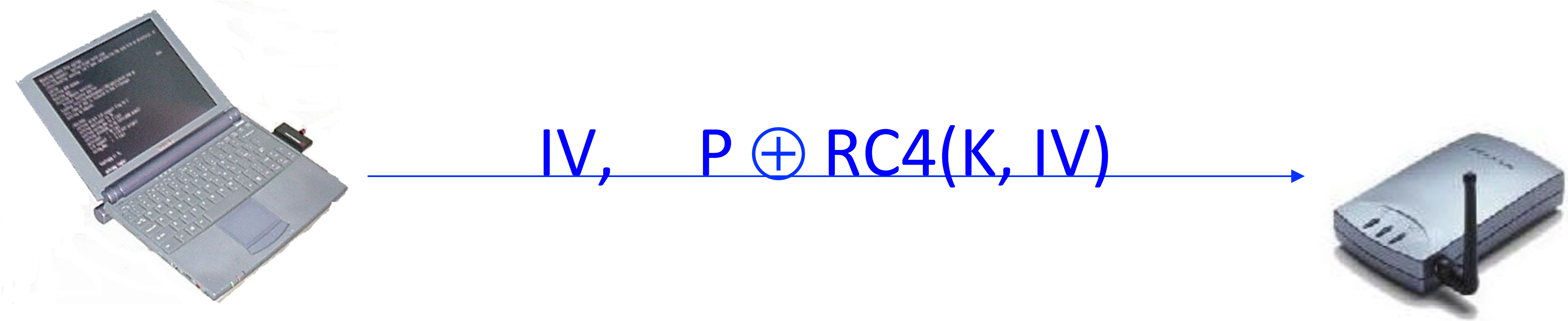
- Re-using a nonce or IV leads to catastrophic security failure.

WEP



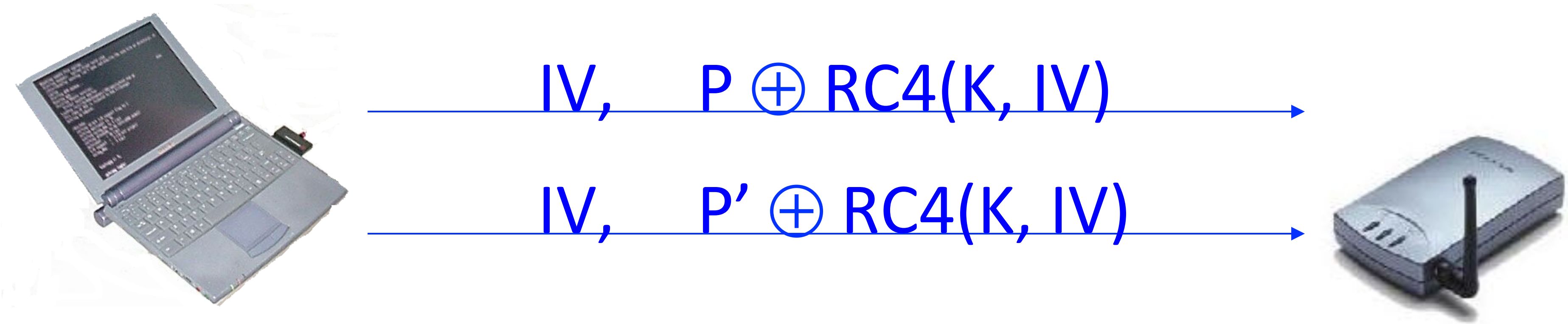
- Early method for encrypting Wifi: WEP (Wired Equivalent Privacy)
  - Share a single cryptographic key among all devices
  - Encrypt all packets sent over the air, using the shared key
  - Use a checksum to prevent injection of spoofed packets

WEP - A Little More Detail



- WEP uses the RC4 stream cipher to encrypt a TCP/IP packet (P) by xor-ing it with keystream (RC4(K, IV))

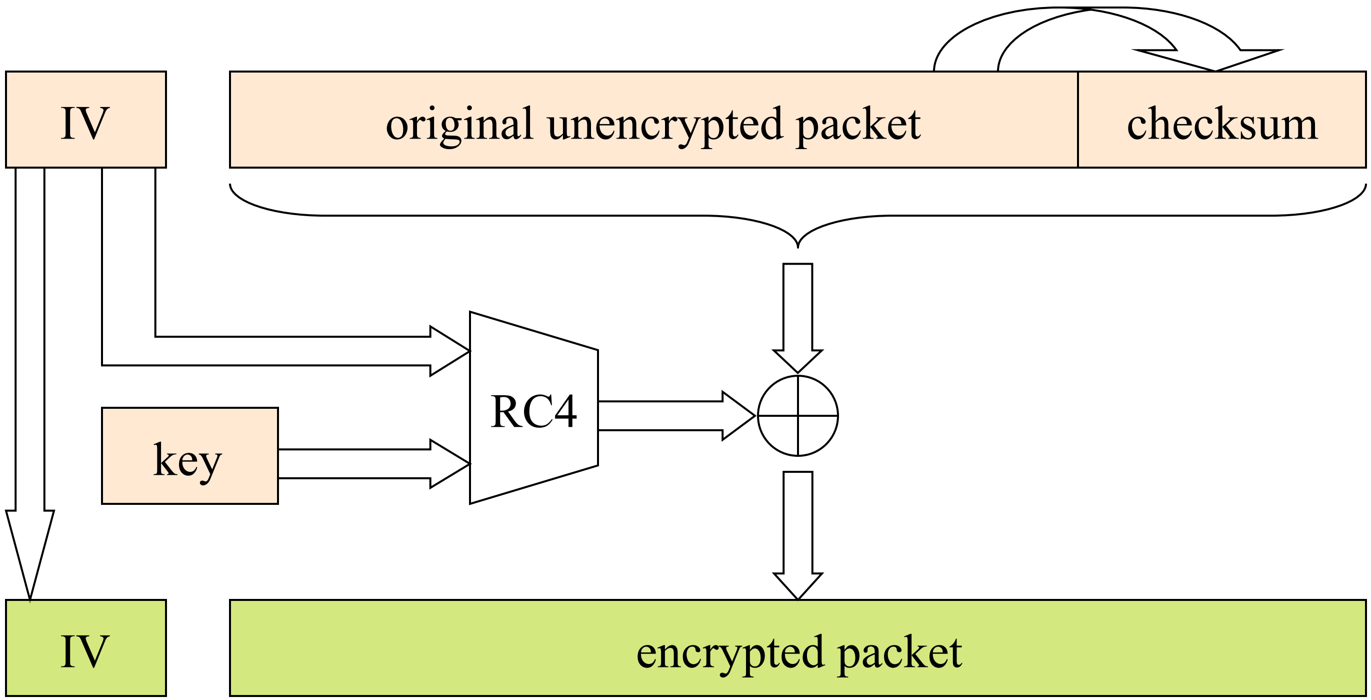
A Risk of Keystream Reuse



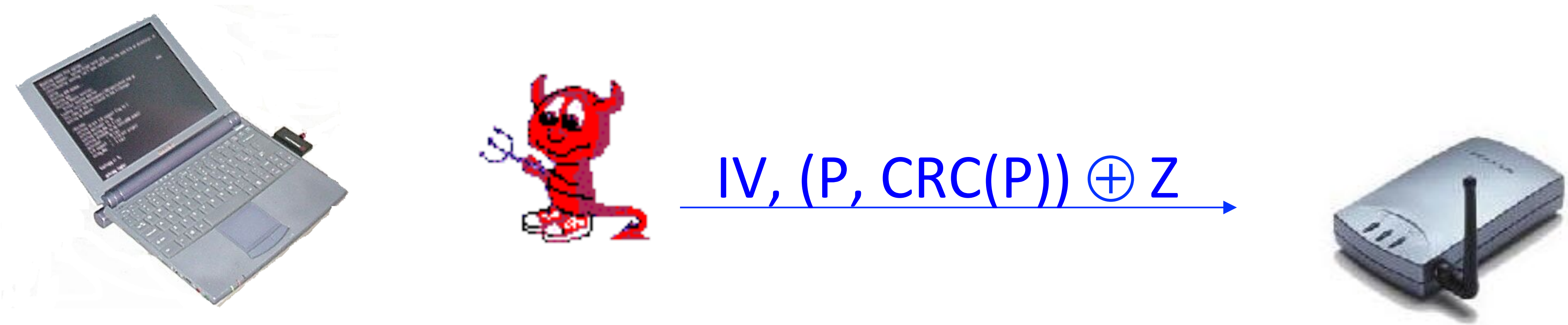
- In some implementations, IVs repeat.
  - If we send two ciphertexts (C, C') using the same IV, then the xor of plaintexts leaks ($P \oplus P' = C \oplus C'$), which might reveal both plaintexts

➤ Lesson: Don't re-use nonces/IVs

WEP -- Even More Detail

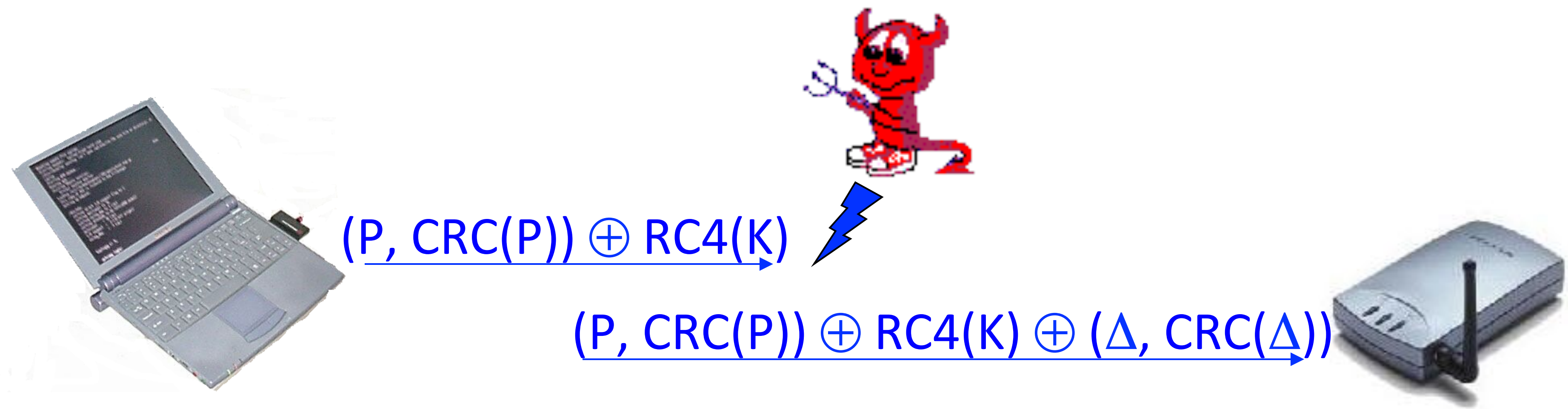


Attack #2: Spoofed Packets



- Attackers can inject forged 802.11 traffic
  - Learn Z = RC4(K, IV) using previous attack
  - Since the CRC checksum is unkeyed, you can then create valid ciphertexts that will be accepted by the receiver

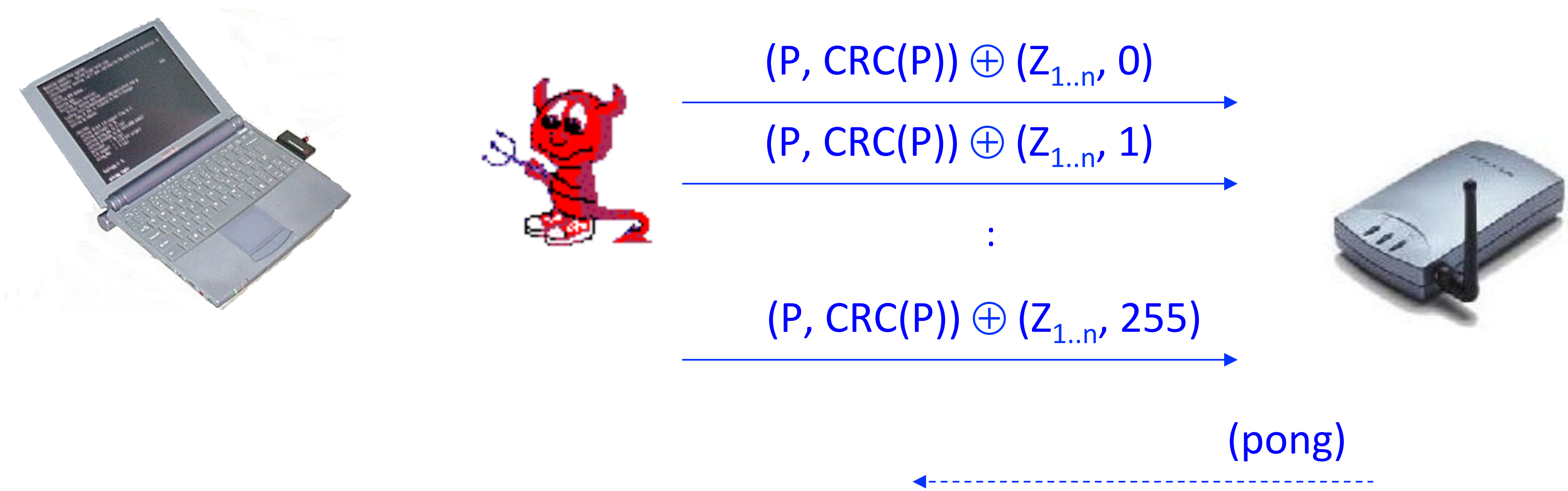
Attack #3: Packet Modification



- CRC is linear
  - ⇒ $CRC(P \oplus \Delta) = CRC(P) \oplus CRC(\Delta)$
    - ⇒ the modified packet ($P \oplus \Delta$) has a valid checksum

➤ Attacker can tamper with packet (P) without breaking RC4

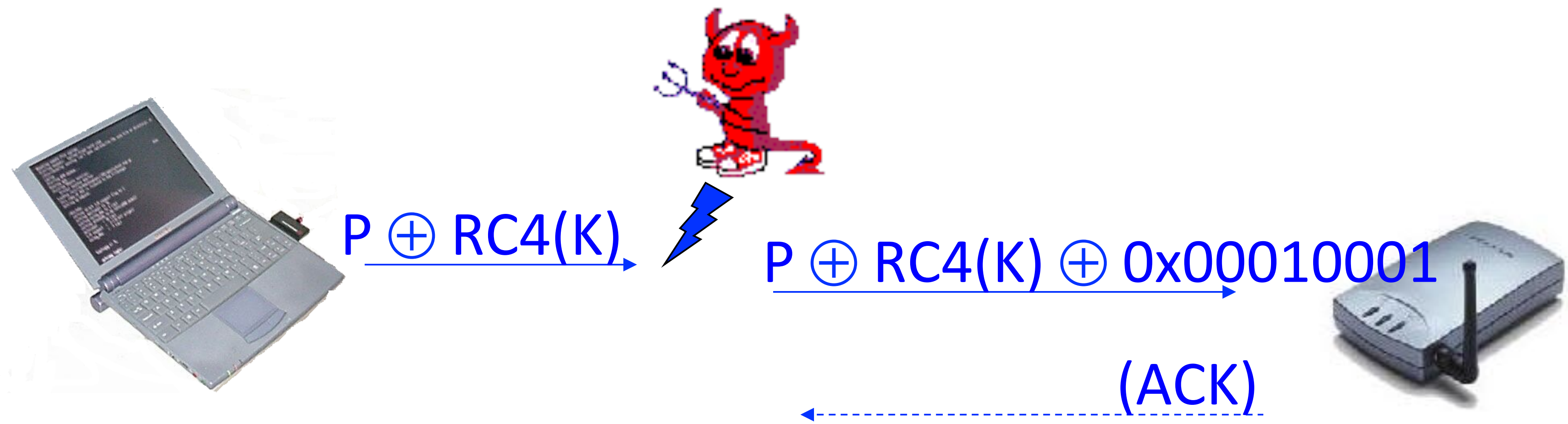
Attack #4: Inductive Learning



- Learn $Z_{1..n}$ = $RC4(K, IV)_{1..n}$ using previous attack
- Then guess Z_{n+1}; verify guess by sending a ping packet ((P, CRC(P))) of length n+1 and watching for a response
- Repeat, for n=1,2,…, until all of RC4(K, IV) is known

Credits: Arbaugh, et al.

Attack #5: Reaction Attacks



- TCP ACKnowledgement returned by recipient
  ⇔ TCP checksum on modified packet (P ⊕ 0x00010001) is valid
  ⇔ wt(P & 0x00010001) = 1

➤ Attacker can recover plaintext (P) without breaking RC4