Protecting TLS from Legacy Crypto

http://mitls.org

Karthikeyan Bhargavan
+ many, many others.
(INRIA, Microsoft Research, LORIA, IMDEA,
Univ of Pennsylvania, Univ of Michigan, JHU)

Agile Cryptographic Protocols

Popular cryptographic protocols *evolve*

- SSL v3 → TLS 1.2
- DH-768 `→ Curve25519
- MD5 → SHA-256

Agility: graceful transition from old to new

- *Negotiate* best shared version, cipher, DH group

What can go wrong?

- We get lazy and forget to remove weak algorithms
- Downgrade attacks that exploit obsolete legacy crypto

Attacks on Legacy Crypto in TLS

• RC4	Keystream biases	[Mar’13]
• Lucky 13	MAC-Encode-Encrypt CBC	[May’13]
• POODLE	SSLv3 MAC-Encode-Encrypt	[Dec’14]
• FREAK	Export-grade 512-bit RSA	[Mar’15]
• LOGJAM	Export-grade 512-bit DH	[May’15]
• SLOTH	RSA-MD5 signatures	[Jan’16]
• DROWN	SSLv2 RSA-PKCS#1v1.5	[Mar’16]

- TLS was supposed to prevent downgrade attacks
- What went wrong? How do we fix it in TLS 1.3?

Transport Layer Security (1994—)

The default secure channel protocol?

HTTPS, 802.1x, VPNs, files, mail, VoIP, ...

20 years of attacks and fixes

1994 Netscape's Secure Sockets Layer
1996 SSLv3
1999 TLS1.0 (RFC2246)
2006 TLS1.1 (RFC4346)
2008 TLS1.2 (RFC5246)
2016? TLS1.3

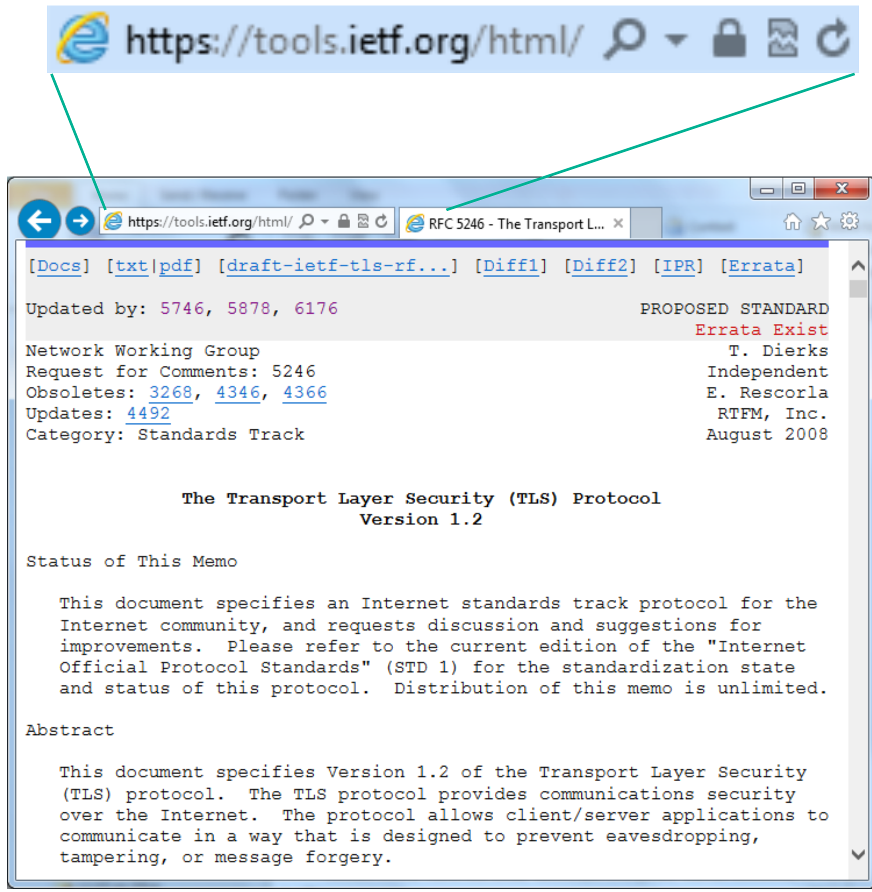
Many implementations

OpenSSL, SecureTransport, NSS, SChannel, GnuTLS, JSSE, PolarSSL, ...

many bugs, attacks, patches every year

Many security theorems

mostly for small simplified models of TLS



[Docs] [txt|pdf] [draft-ietf-tls-rf...] [Diff1] [Diff2] [IPR] [Errata]

Updated by: 5746, 5878, 6176 PROPOSED STANDARD
Errata Exist

Network Working Group T. Dierks
Request for Comments: 5246 Independent
Obsoletes: 3268, 4346, 4366 E. Rescorla
Updates: 4492 RTFM, Inc.
Category: Standards Track August 2008

The Transport Layer Security (TLS) Protocol
Version 1.2

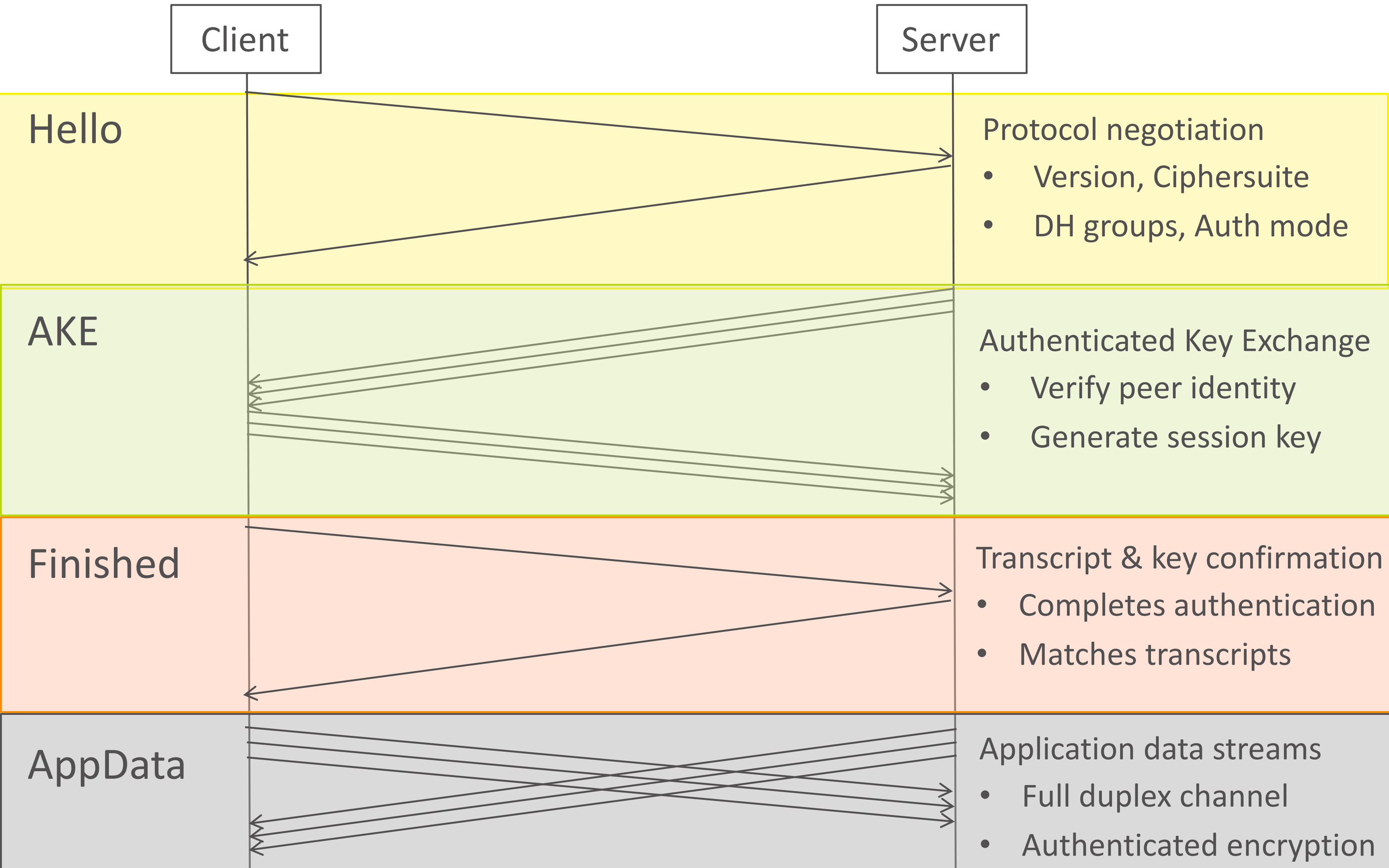
Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

TLS protocol overview



Protocol Agility in TLS

Protocol versions

- TLS 1.2, TLS 1.1, TLS 1.0, SSLv3, SSLv2

Key exchanges

- ECDHE, FFDHE, RSA, PSK, …

Authentication modes

- ECDSA, RSA signatures, PSK,…

Authenticated Encryption Schemes

- AES-GCM, CBC MAC-Encode-Encrypt, RC4,…

100s of possible protocol combinations!

Example Protocol Instance

TLS_RSA_WITH_AES_128_CBC_SHA

RSA Key Transport	AES-CBC + HMAC
• RSA-PKCS#1 v1.5 encryption	• MAC-Encode-Encrypt Scheme
• [1998] Bleichenbacher attack and fixes	• [2002] Vaudenay attack
• [2013] Crypto proof for TLS-RSA	• [2011] Crypto proof for TLS MEE-CBC
• [2016] DROWN attack: downgrade to SSLv2	• [2013] Lucky 13 attack
	• [2014] Poodle attack on SSLv3
	• [2016] Verified implementation

The Modeling Gap

Textbook crypto proofs not applicable to TLS

- It uses classic constructs in non-standard ways
- Needs protocol-specific assumptions and proofs
- Much recent progress: sLHAE, ACCE, miTLS

Theoretical attacks not always exploitable

- Attack may be thwarted by protocol details
- Practitioners only respond to practical attacks
- Leads to a communication gap between cryptographers and practitioners

The Protocol Composition Gap

Most crypto proofs are for single constructs

- TLS-DHE, TLS-RSA, TLS-PSK, MEE-CBC

Many attacks appear only in composition

- Downgrades and cross-protocol attacks
- State-machine flaws in implementations

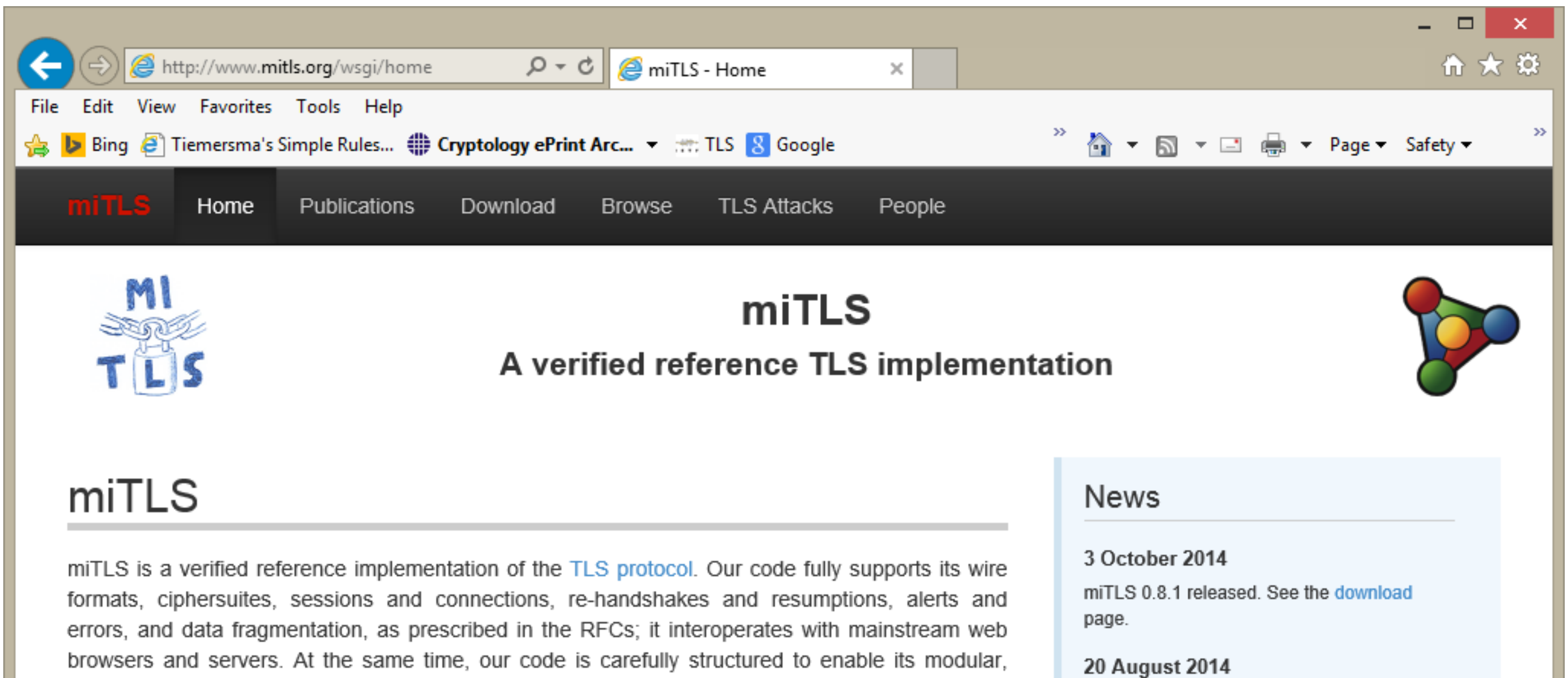
Too many compositions to prove by hand

- We need automated verification tools that can analyze both protocols and implementations

miTLS: Closing the Gap

A verified reference implementation of TLS

- Covers TLS 1.0-1.2, dozens of ciphersuites
- Accounts for messy low-level protocol details



miTLS: New TLS Attacks

Triple Handshake Attacks [S&P 2014]

- Breaking client authentication by composing three different handshake modes

State Machine Attacks (e.g. FREAK) [S&P 2015]

- Bugs in the composite state machines implemented by mainstream TLS libraries

Logjam [CCS 2015]

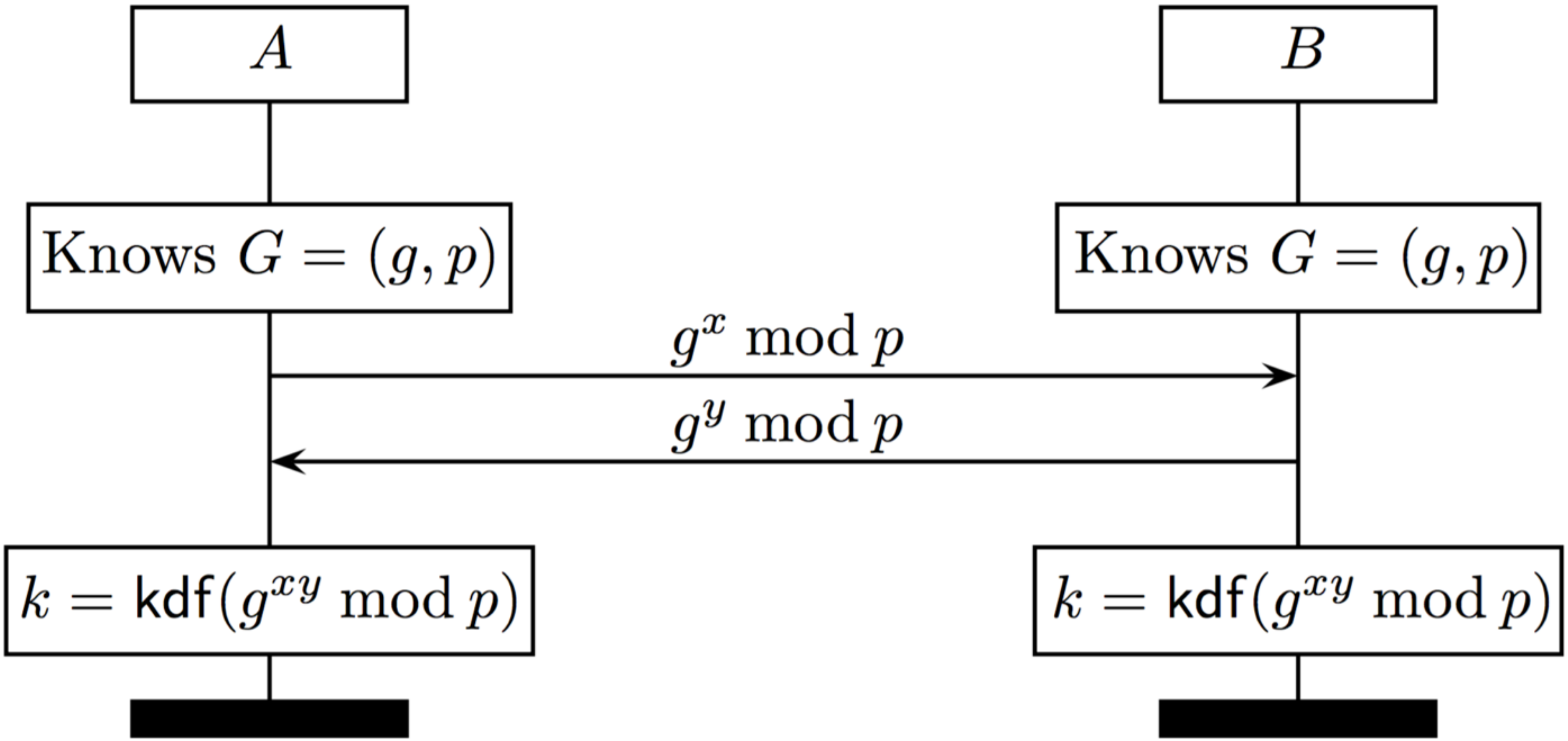
- DH group downgrade using DHE_EXPORT

SLOTH [NDSS 2016]

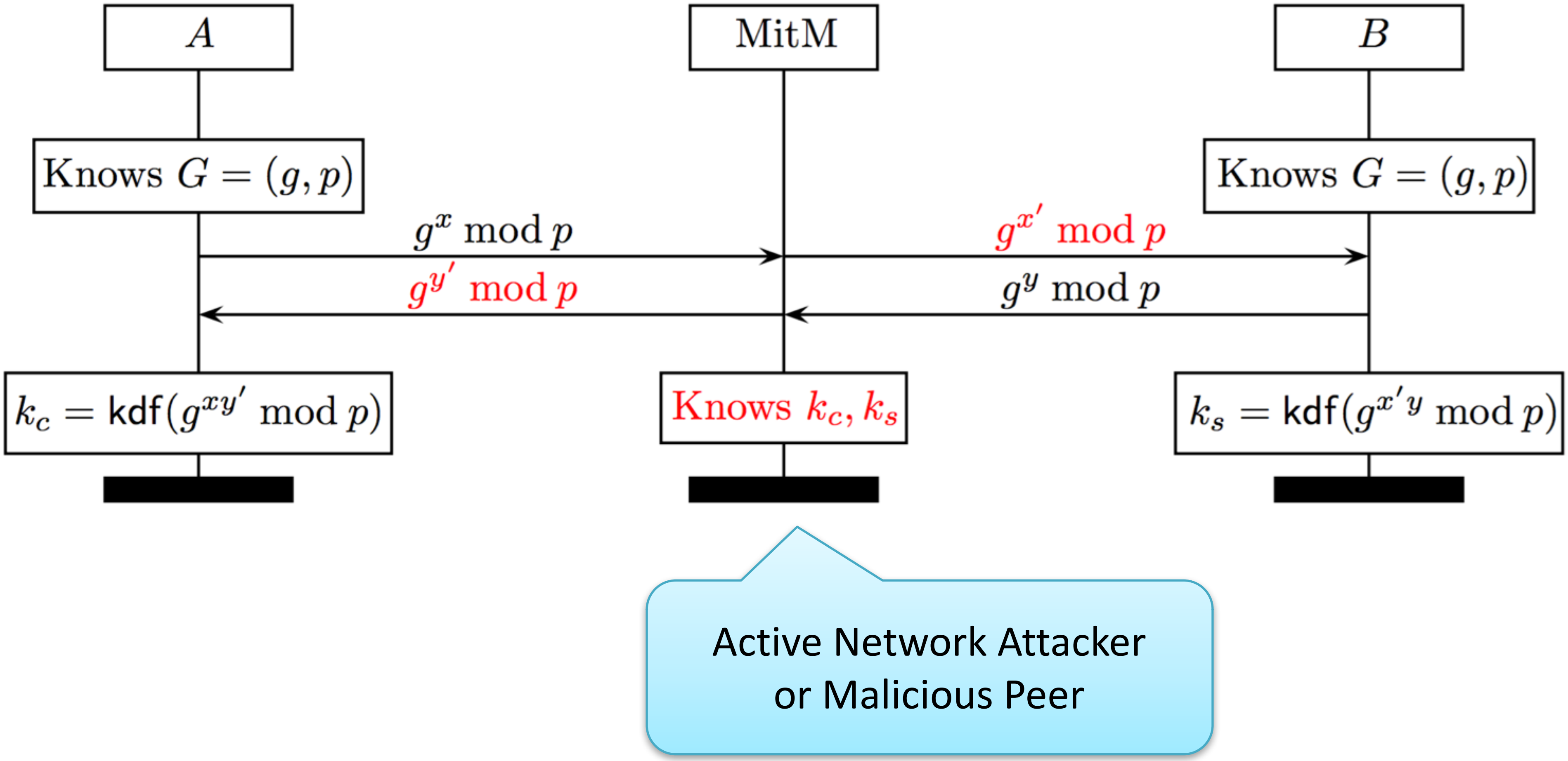
- Hash function downgrade for transcript collisions

Downgrade Attacks on Agile Key Exchange

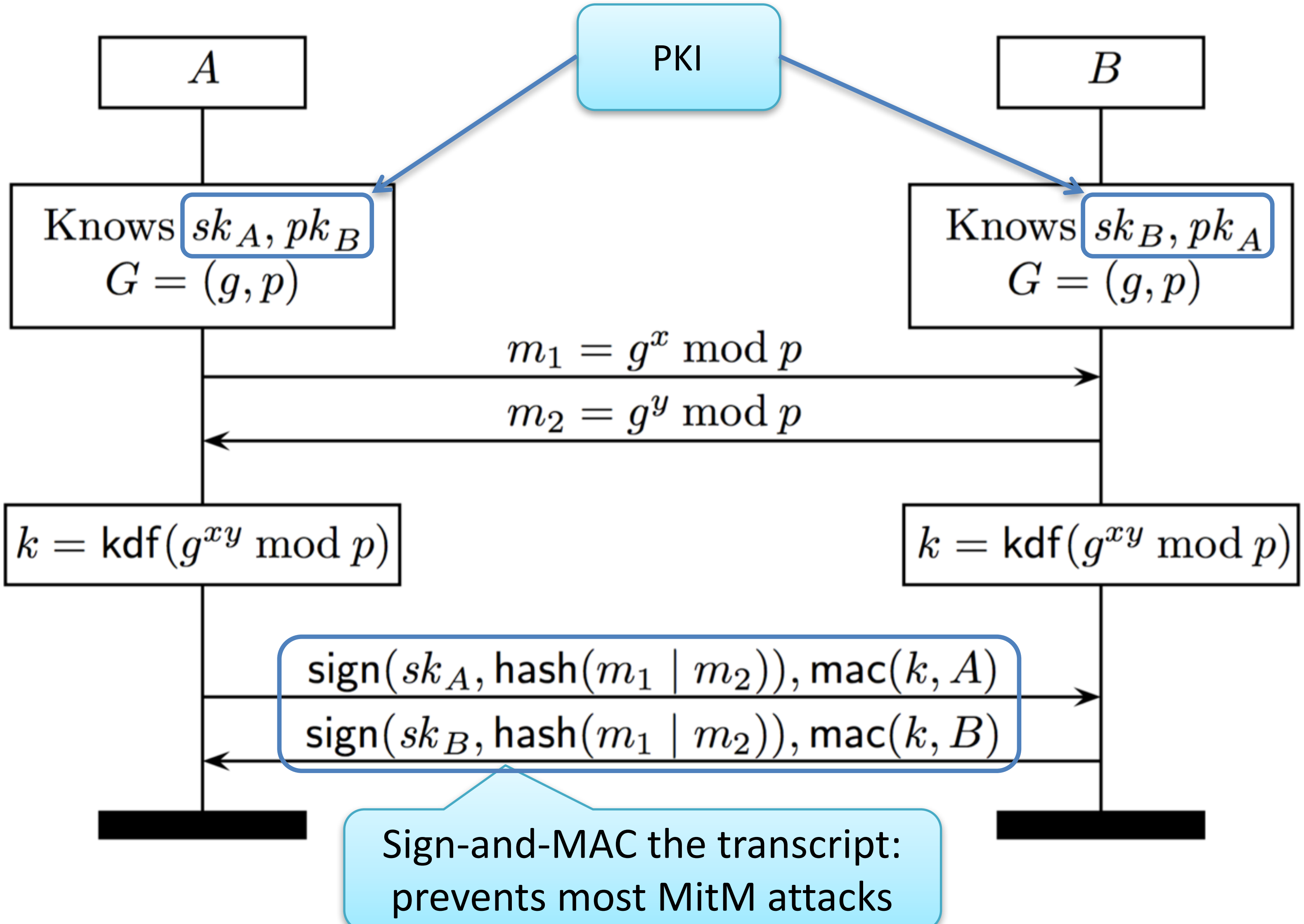
Anonymous Diffie-Hellman (DH_{anon})



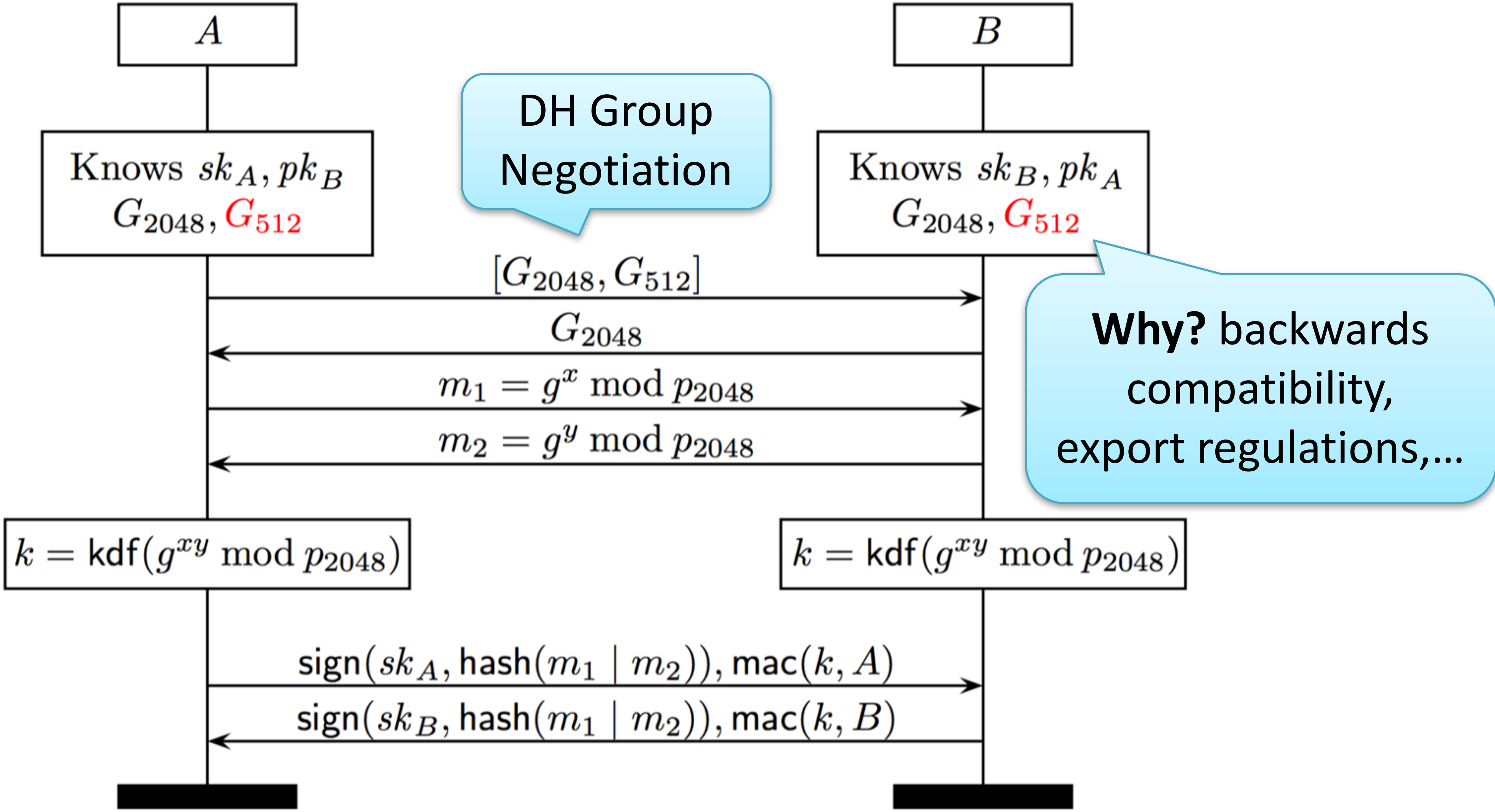
Man-in-the-Middle attack on DH_{anon}



SIGMA: Authenticated DH



SIGMA with Group Negotiation



Export-Grade 512-bit DHE in TLS

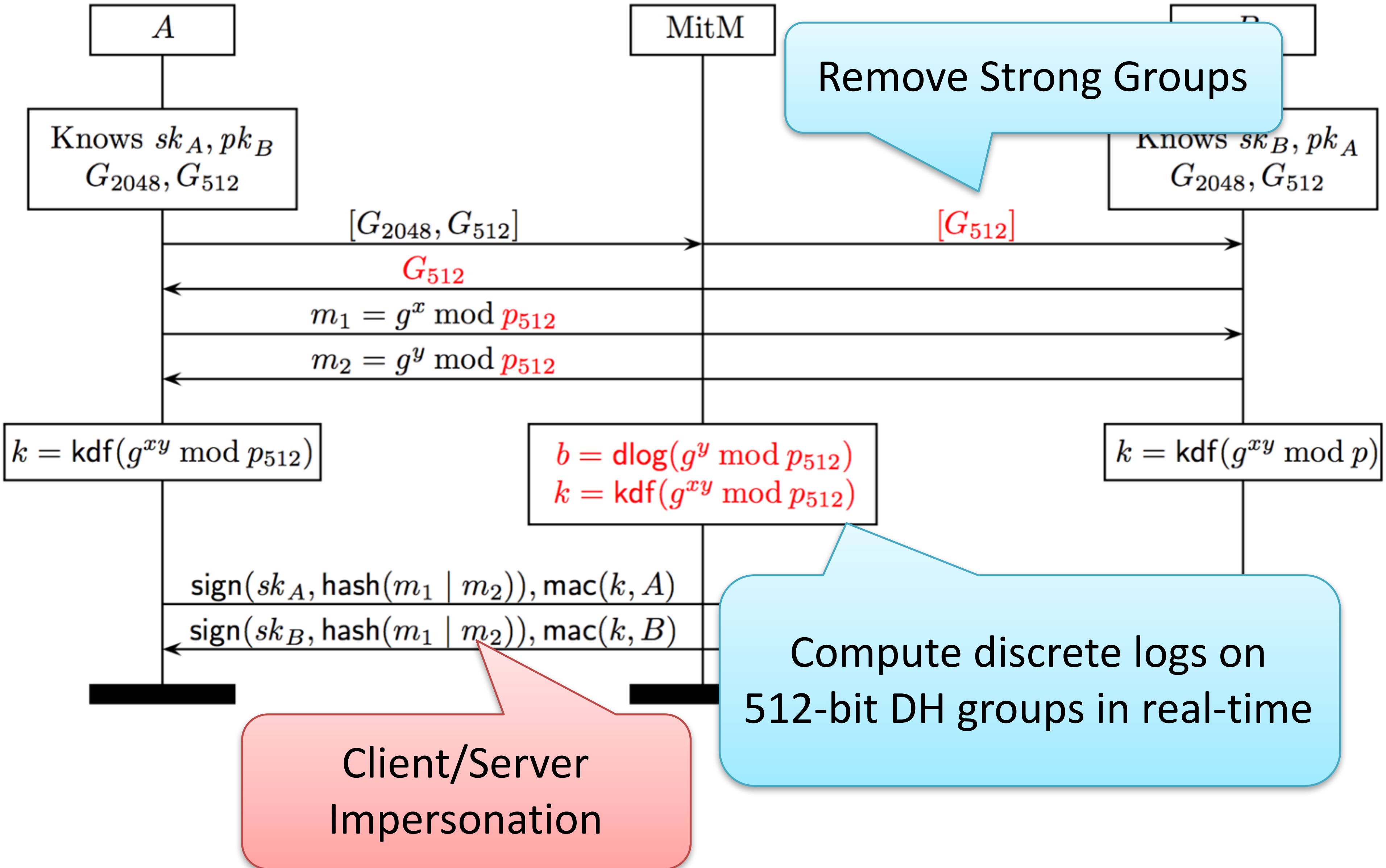
TLS 1.0 supported deliberately weakened ciphers to comply with export regulations in 1990s

- DH groups/RSA keys limited to 512 bits

EXPORT deprecated in 2000, but still supported by TLS in 2015

- 8.4% of Top 1M websites in March 2015
- Browsers only support DHE, not DHE_EXPORT but will accept 512-bit DH groups for DHE
- Protocol flaw:
  Server's DHE and DHE_EXPORT key-shares and signatures look the same to a TLS client

Logjam: MitM Group Downgrade Attack



Downgrade Protection in TLS 1.2

- In TLS 1.2, both client and server MAC the full transcript to prevent tampering:

$$\textbf{mac}(k, [G_{2048}, G_{512}] \mid G_{512} \mid m_1 \mid m_2)$$

- But it's too late, we already used G_{512} to compute k

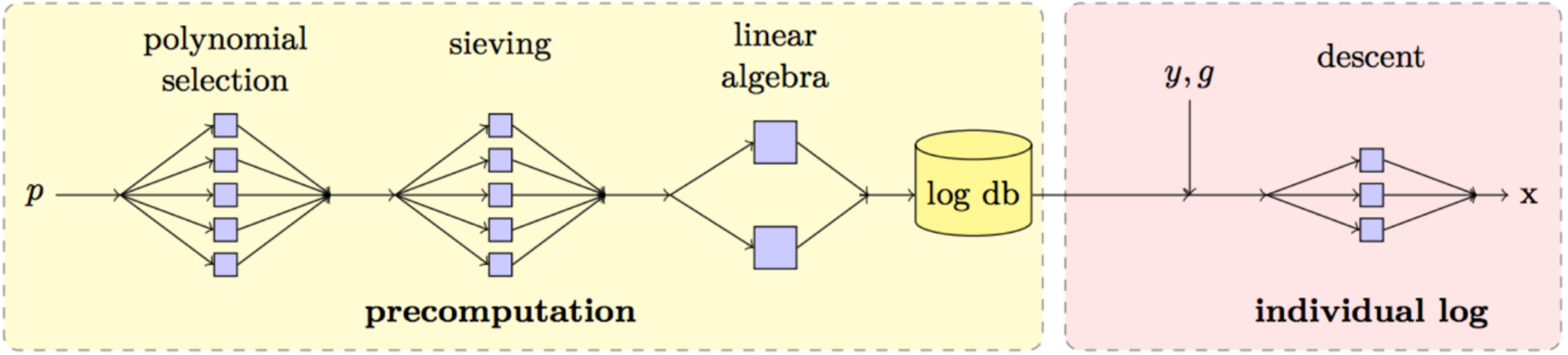
$$k = \textbf{kdf(}g^{xy} \bmod p_{512})$$

so, the attacker can compute k and forge the MAC

The TLS 1.2 downgrade protection mechanism itself depends on downgradeable parameters!

- We can break it if we can compute the discrete log while the connection is still live

Logjam: Exploiting Pre-Computation



Most TLS servers use well-known 512-bit groups

- 92% of DHE_EXPORT servers use one of two groups
- 1-2 weeks of precomputation per group (CADO-NFS)
- 90 seconds to compute discrete log for each key
- Practitioners seemingly unaware of this optimization!

Logjam: Impact and Countermeasures

The TLS transcript MAC does not prevent Diffie-Hellman group downgrades

- Must disable all weak DH groups and elliptic curves
- Browsers moving to 1024-bit minimum group size
- Breaking 768-bit and 1024-bit groups will have a catastrophic impact on TLS, SSH, and IPsec

Could we do better by relying on transcript signatures for downgrade protection?

Downgrade Protection via Signatures

IKEv1: both A and B sign the offered groups

- **sign**(sk_B, **hash**($[G_{2048}, G_{512}] \mid m_1 \mid m_2$))
- no agreement on chosen group!

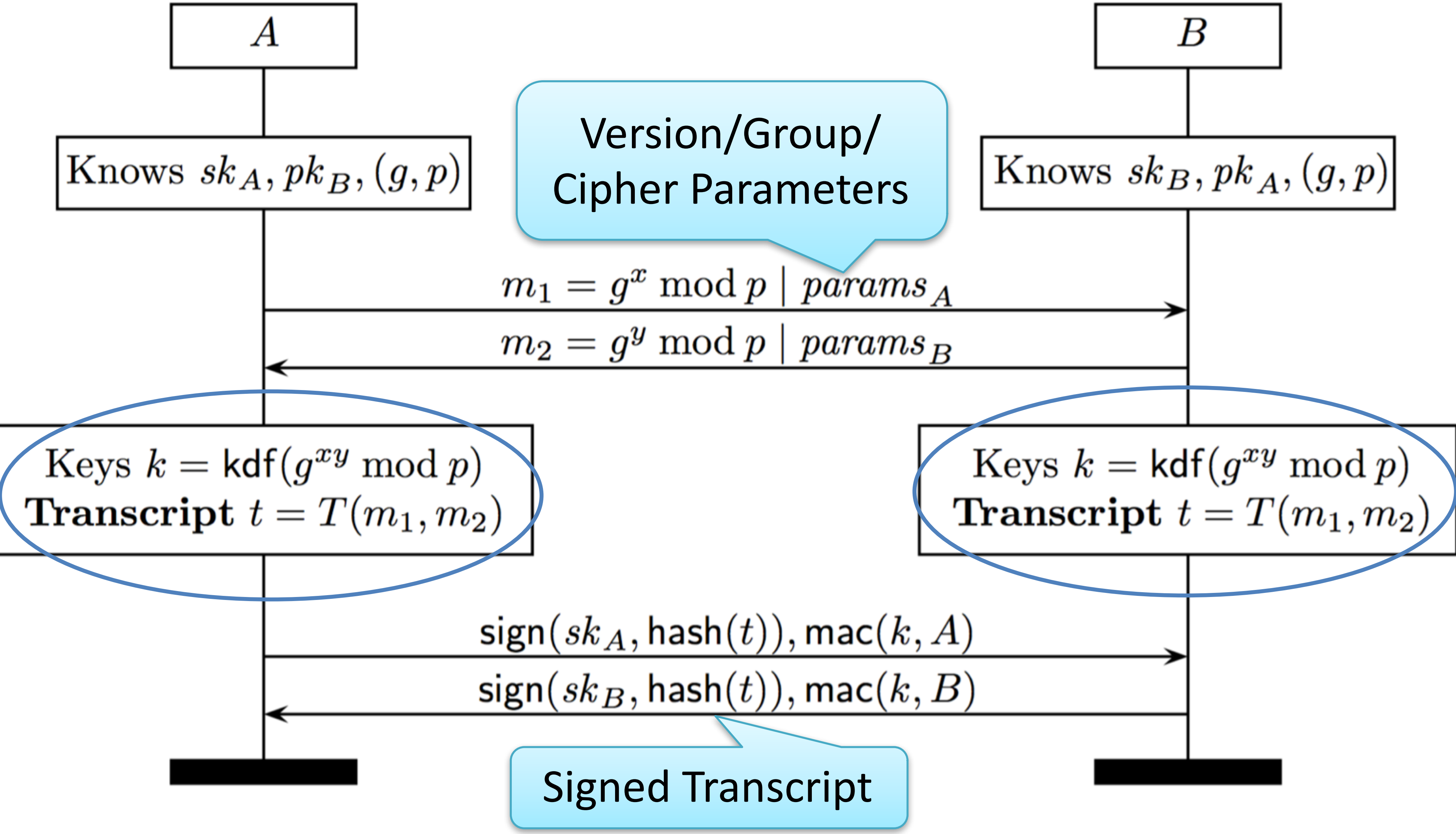
IKEv2: each party signs its own messages

- **sign**(sk_A, **hash**($[G_{2048}, G_{512}] \mid m_1$))
- **sign**(sk_B, **hash**($G_{512} \mid m_2$))
- no agreement on offered groups!

SSH-2 and **TLS 1.3**: sign the full transcript

- **sign**(k, **hash**($[G_{2048}, G_{512}] \mid G_{512} \mid m_1 \mid m_2$))
- Prevents Logjam (but what about other downgrades?)

SIGMA with Generic Negotiation

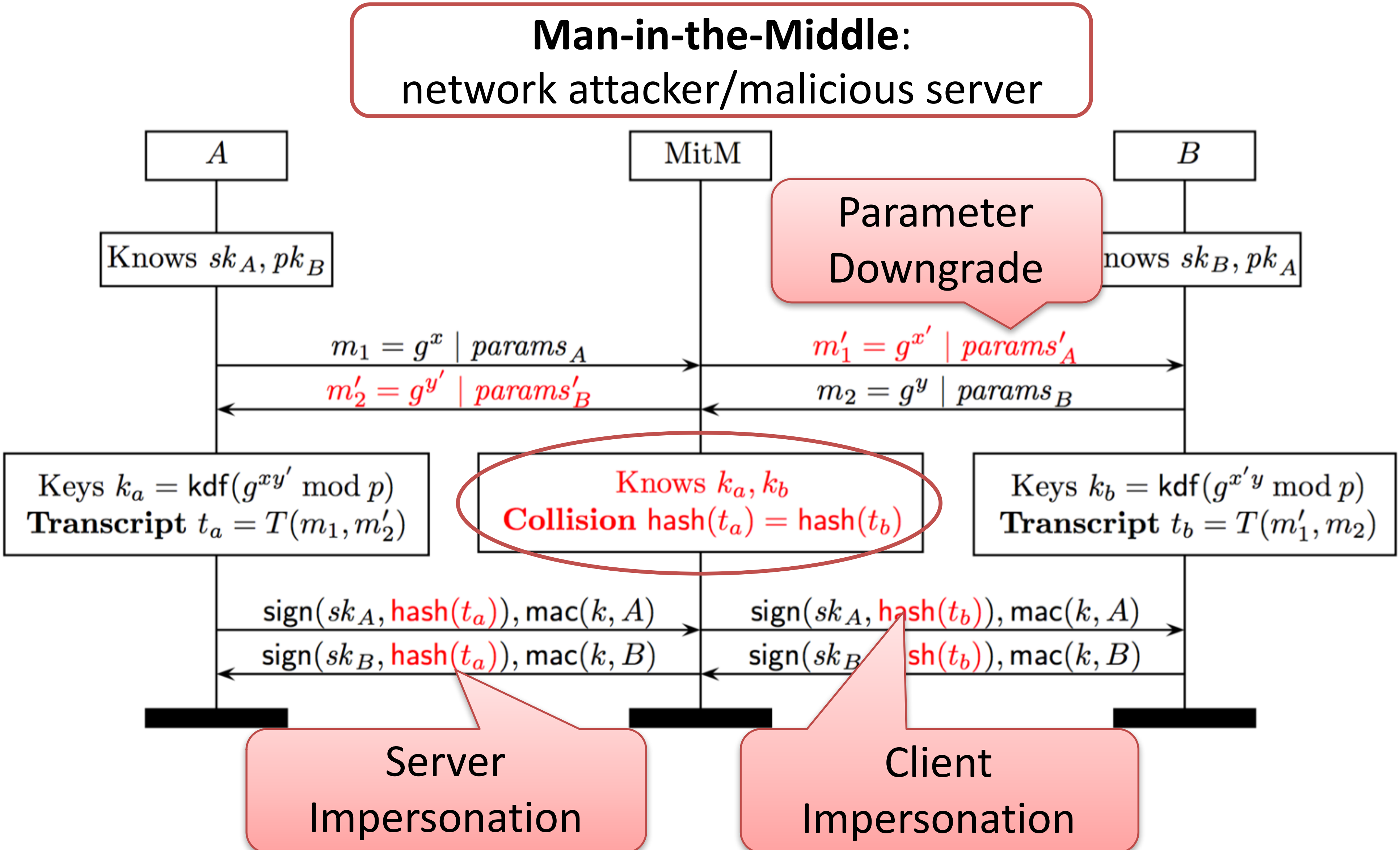


Downgrade Protection via Signatures

- Sign the full transcript
  - **sign**(sk_B, **hash**($m_1 \mid m_2$))
  - *Example*: TLS 1.3, SSH-2, TLS 1.2 client auth

- How weak can the **hash** function be?
  - do we need collision resistance?
  - do we only need 2nd preimage resistance?
  - Is it still safe to use MD5, SHA-1 in TLS, IKE, SSH?
  - *Disagreement*: cryptographers vs. practitioners
    (see Schneier vs. Hoffman, RFC4270)

SLOTH: Transcript Collision Attacks



Computing a Transcript Collision

$$\mathbf{hash}(m_1 \mid m'_2) = \mathbf{hash}(m'_1 \mid m_2)$$

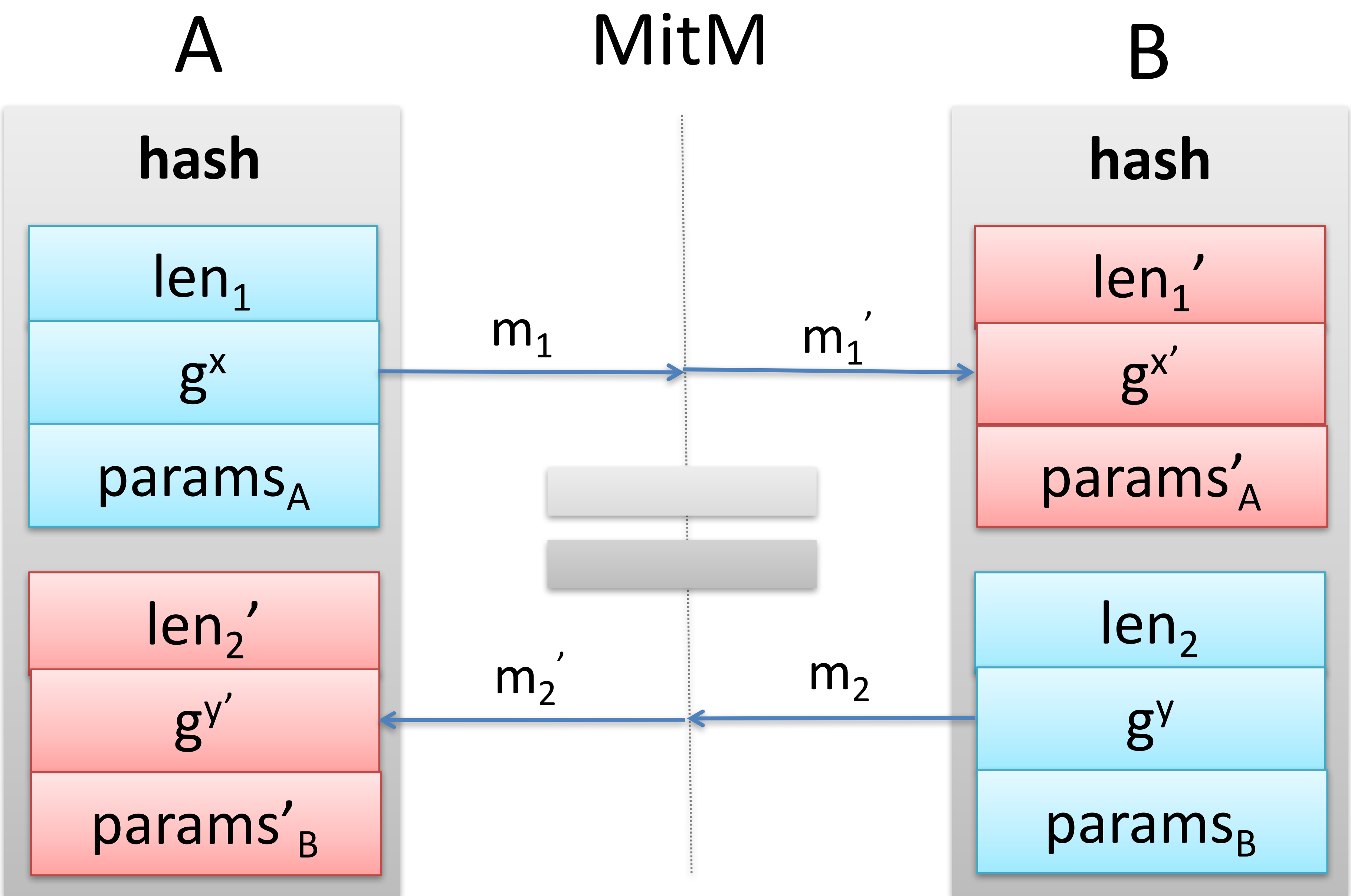
- We need to compute a collision, *not a preimage*
  - Attacker controls parts of both transcripts
  - If we know the black bits, can we compute the red bits?
  - This can sometimes be set up as a **generic collision**
- If we're lucky, we can set up a **shortcut** collision
  - **Common-prefix**: collision after a shared transcript prefix
  - **Chosen-prefix:** collision after attacker-controlled prefixes

Primer on Hash Collision Complexity

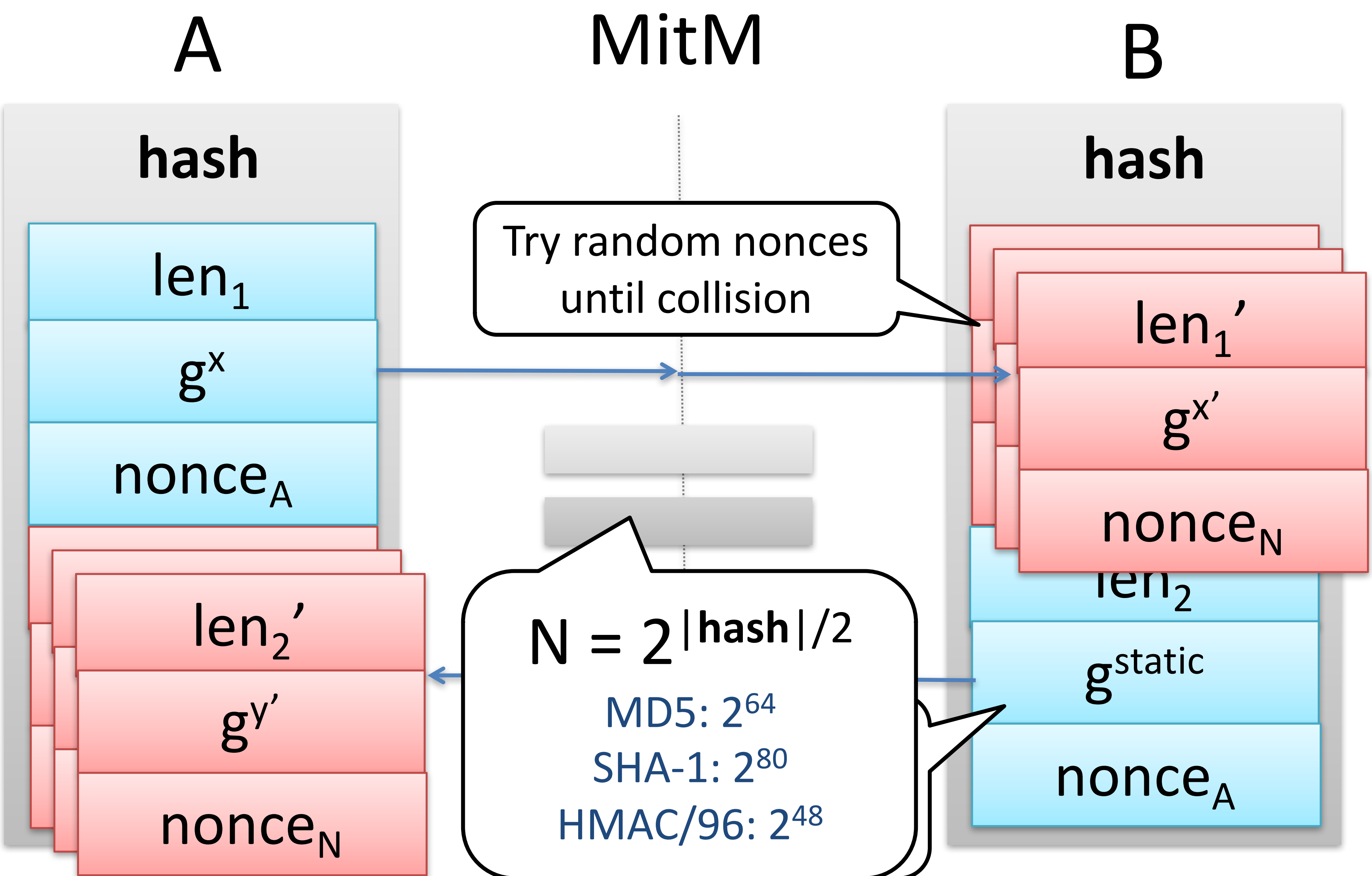
- MD5: known attack complexities
  - **MD5** second preimage 2^{128} hashes
  - **MD5** generic collision: 2^{64} hashes (birthday)
  - **MD5** chosen-prefix collision: 2^{39} hashes (1 hour)
  - **MD5** common-prefix collision: 2^{16} hashes (seconds)

- SHA1: estimated attack complexities
  - **SHA1** second preimage 2^{160} hashes
  - **SHA1** generic collision: 2^{80} hashes (birthday)
  - **SHA1** chosen-prefix collision: 2^{77} hashes (?)
  - **SHA1** common-prefix collision: 2^{61} hashes (?)

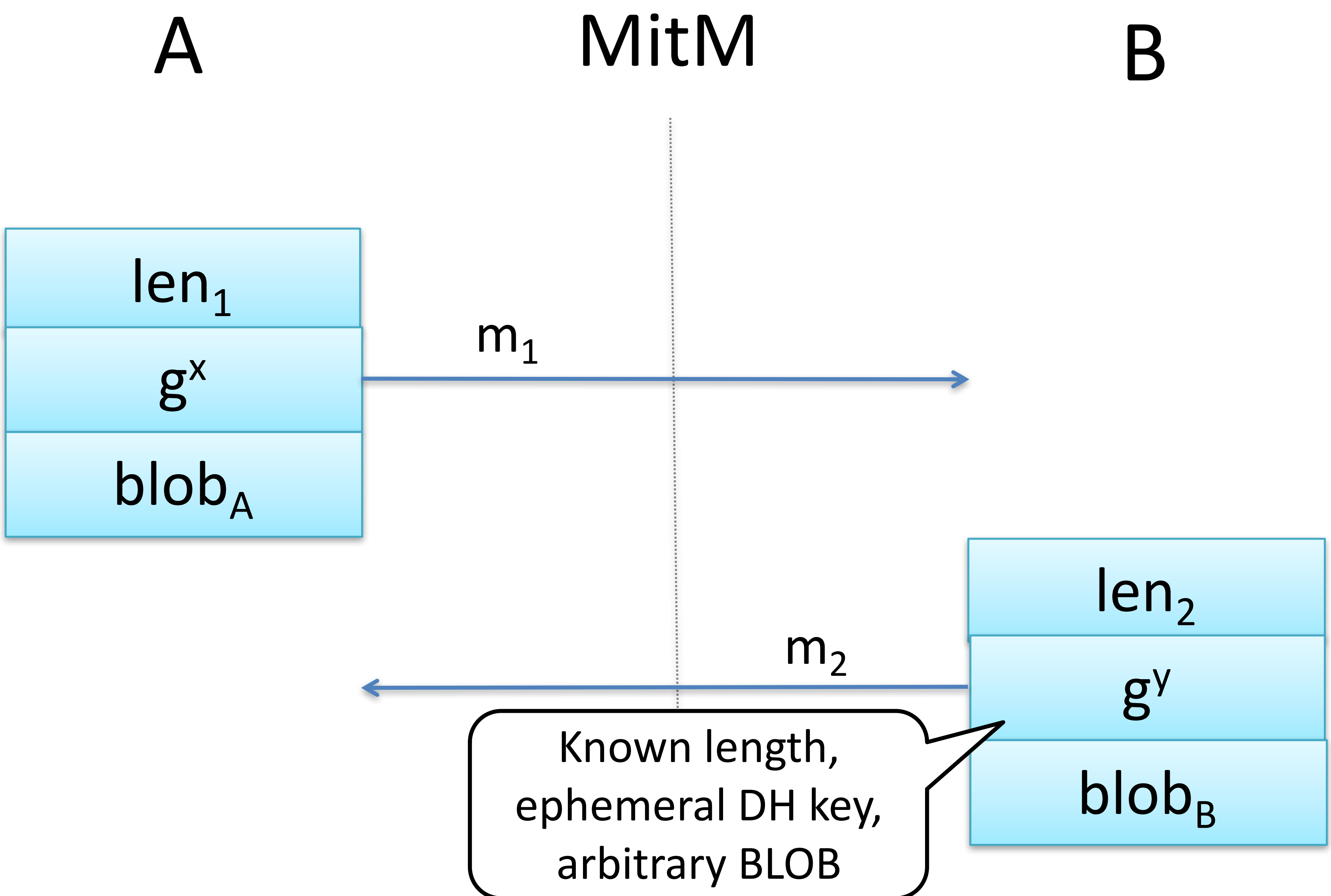
Computing Transcript Collisions

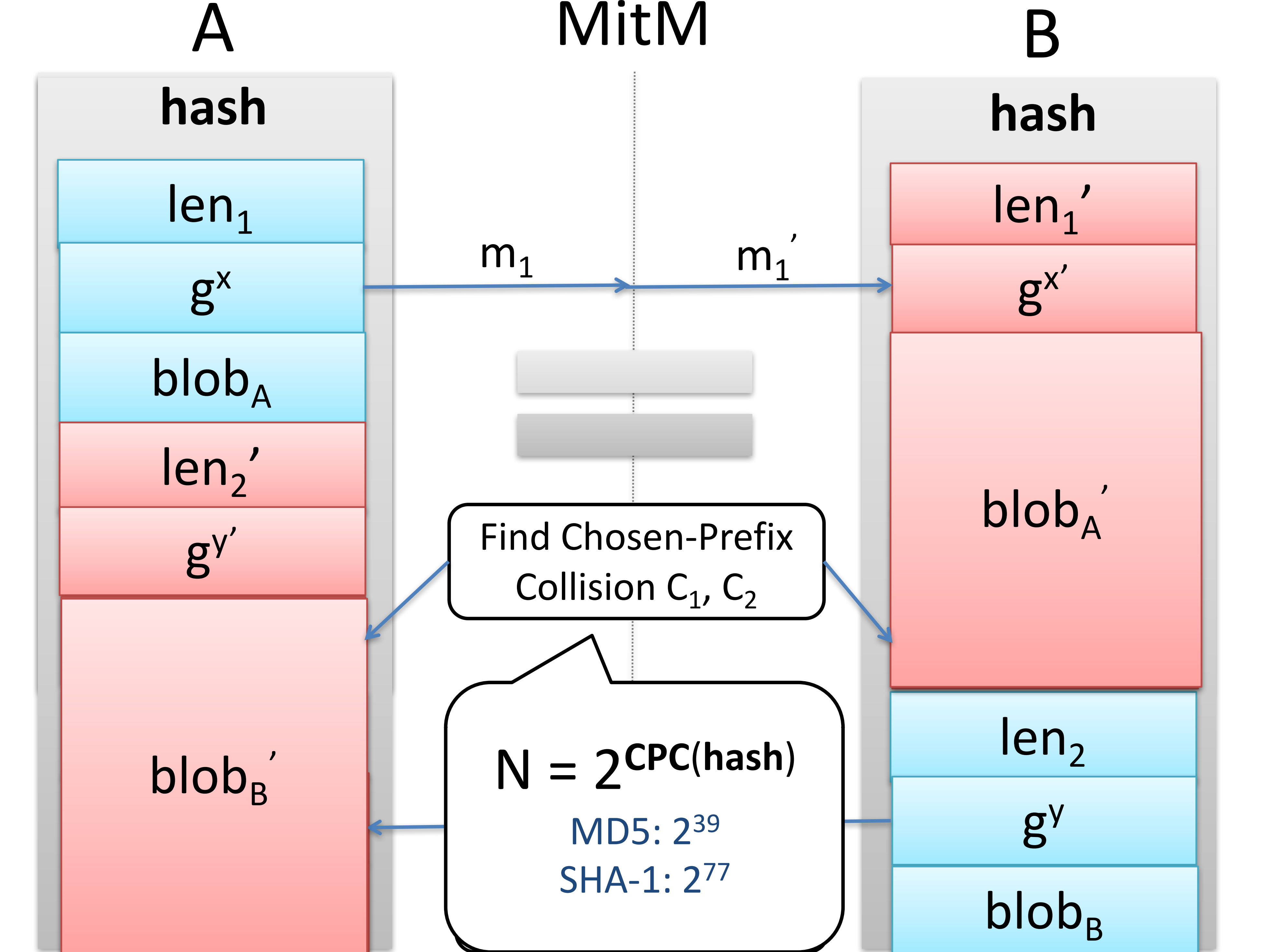


Generic Transcript Collisions



Chosen-Prefix Transcript Collisions





A
MitM
B
hash
len1
gx
blobA
len2'
gy'
blobB'
m1
m1'
hash
len1'
gx'
blobA'
len2
gy
blobB
Find Chosen-Prefix
Collision C1, C2
N = 2CPC(hash)
MD5: 239
SHA-1: 277

Downgrading and Attacking TLS 1.2

TLS 1.2 upgraded the hash functions used in TLS

- TLS 1.1 hard-coded the use of MD5 || SHA-1
- TLS 1.2 uses SHA-256 for all handshake constructions
- Allows negotiation of hash functions: SHA-256/384/512

TLS 1.2 added support for MD5-based signatures!

- Even if the client and server prefer **RSA-SHA256**, the connection can be downgraded to RSA-MD5!

Transcript collisions break TLS 1.2 client signatures

- Chosen prefix collision exploiting flexible message formats
- **Demo:** Takes 1 hour/connection on a 48-core workstation
- *Not very practical*: connection must be live during attack

Attacking TLS Server Auth

- TLS 1.2 server signatures are harder to break
  - *Irony*: the weakness that enables Logjam blocks SLOTH
  - Needs 2^X prior connections + 2^{128-X} hashes/connection
  - Not practical for academics, as far as we know

- TLS 1.3 server signatures is potentially vulnerable
  - *New*: MD5, SHA-1 sigs now explicitly forbidden in TLS 1.3

Other Hash Constructions in TLS

- When used as **transcript hash functions** many constructions are not collision resistant
  - **MD5**(x) | **SHA1**(x)
    not much better than SHA1
  - **HMAC-MD5**(k,x)
    not much better than MD5
  - **HMAC-SHA256**(k,MD5(x))
    not much better than MD5
  - **Truncated HMAC-SHA256**(k,x) to N bits
    not much better than a N bit hash function

Other SLOTH Vulnerabilities

Reduced security for TLS 1.*, IKEv1, IKEv2, SSH

- Impersonation attack on TLS channel bindings
- Exploits downgrades + transcript collisions
- Protocol flaws, not implementation bugs
- Only mitigation is to disable weak hash functions

Protocol	Property	Mechanism	Attack	Collision Type	Precomp.	Work/conn.	Preimage	Wall-clock time
TLS 1.2	Client Auth	RSA-MD5	Impersonation	Chosen Prefix		2^{39}	2^{128}	48 core hours
TLS 1.3	Server Auth	RSA-MD5	Impersonation	Chosen Prefix		2^{39}	2^{128}	48 core hours
TLS 1.0-1.2	Channel Binding	HMAC (96 bits)	Impersonation	Generic		2^{48}	2^{96}	80 GPU days
TLS 1.2	Server Auth	RSA-MD5	Impersonation	Generic	2^{X} conn.	2^{128-X}	2^{128}	
TLS 1.0-1.1	Handshake Integrity	MD5 | SHA-1	Downgrade	Chosen Prefix		2^{77}	2^{160}	
IKE v1	Initiator Auth	HMAC-MD5	Impersonation	Generic		2^{65}	2^{128}	
IKE v2	Initiator Auth	RSA-SHA-1	Impersonation	Chosen Prefix	2^{77}	0	2^{160}	
SSH-2	Exchange Integrity	SHA-1	Downgrade	Chosen Prefix		2^{77}	2^{160}	

Logjam and SLOTH: Lessons Learned

Legacy crypto can remain hidden for a long time

- Finding DHE_EXPORT, RSA-MD5 enabled was surprising

Important to demonstrate concrete attacks, not just theoretical weaknesses

- Concrete attacks can help motivate new cryptanalytic optimizations, and justify implicit proof assumptions

TLS 1.2 does not prevent some downgrades

- Need for a formal model of downgrade resilience and a new protocol that provably achieves it

Downgrade Resilience in Key Exchange Protocols

AKEs with Parameter Negotiation

- Let’s consider two party protocols (*I* → *R*)
- Key exchange inputs:
  - $config_I$ & $config_R$: supported versions, ciphers, etc.
  - $creds_I$ & $creds_R$: long-term private keys
- Key exchange outputs:
  - *uid*: unique session identifier
  - *k*: session key
  - *mode*: negotiated version, cipher, etc.

Agile AKE Security Goals

- ***Partnering***
  at most one honest partner exists with same *uid*
- ***Agreement***
  if my negotiated *mode* uses only strong algorithms,
  then my partner and I agree on *k* and *mode*
- ***Confidentiality***
  if my negotiated *mode* uses only strong algorithms,
  the key *k is* only known to me and my partner
- ***Authenticity***
  if my intended peer is authenticated and honest,
  and my negotiated *mode* uses only strong algorithms,
  then at least one partner with same *uid* exists

Agile Agreement vs. Downgrades

- ***Agreement***
  if my negotiated *mode* uses only strong algorithms,
  then my partner and I agree on *k* and *mode*

- Agreement does not guarantee that the protocol will negotiate a strong mode
  - So, it does not forbid downgrade attacks
  - To prevent downgrades, all algorithms in the **intersection** of $config_I$ & $config_R$ must be strong
  - What if $config_I$ & $config_R$ include a legacy algorithm ?

A New Downgrade Resilience Goal

- ***Ideal Negotiation:*** *$Nego(config_I, config_R)$*
  Informally, the *mode* that would have been negotiated in the absence of an attacker

- ***Downgrade Resilience***
  The protocol should negotiate the *ideal* mode even in the presence of the attacker

$$mode = Nego(config_I, config_R)$$

(Details in IEEE S&P 2016, see: mitls.org)

Testing the Definition

- ***IKEv1*** does not prevent downgrades
  - Known DH group, ciphersuite downgrades
- ***IKEv2*** does not prevent downgrades
  - New attack on EAP mode
- ***ZRTP*** does not prevent downgrades
  - New attack on pre-shared mode

- ***SSHv2*** is downgrade resilient if SHA-1 not used
  - Stronger agreement theorem than previous work

A new protocol: TLS 1.3

Stronger key exchanges, fewer options

- ECDHE and DHE by default, no RSA key transport
- Strong DH groups (> 2047 bits) and EC curves (> 255 bits)
- Only AEAD ciphers (AES-GCM), no CBC, no RC4

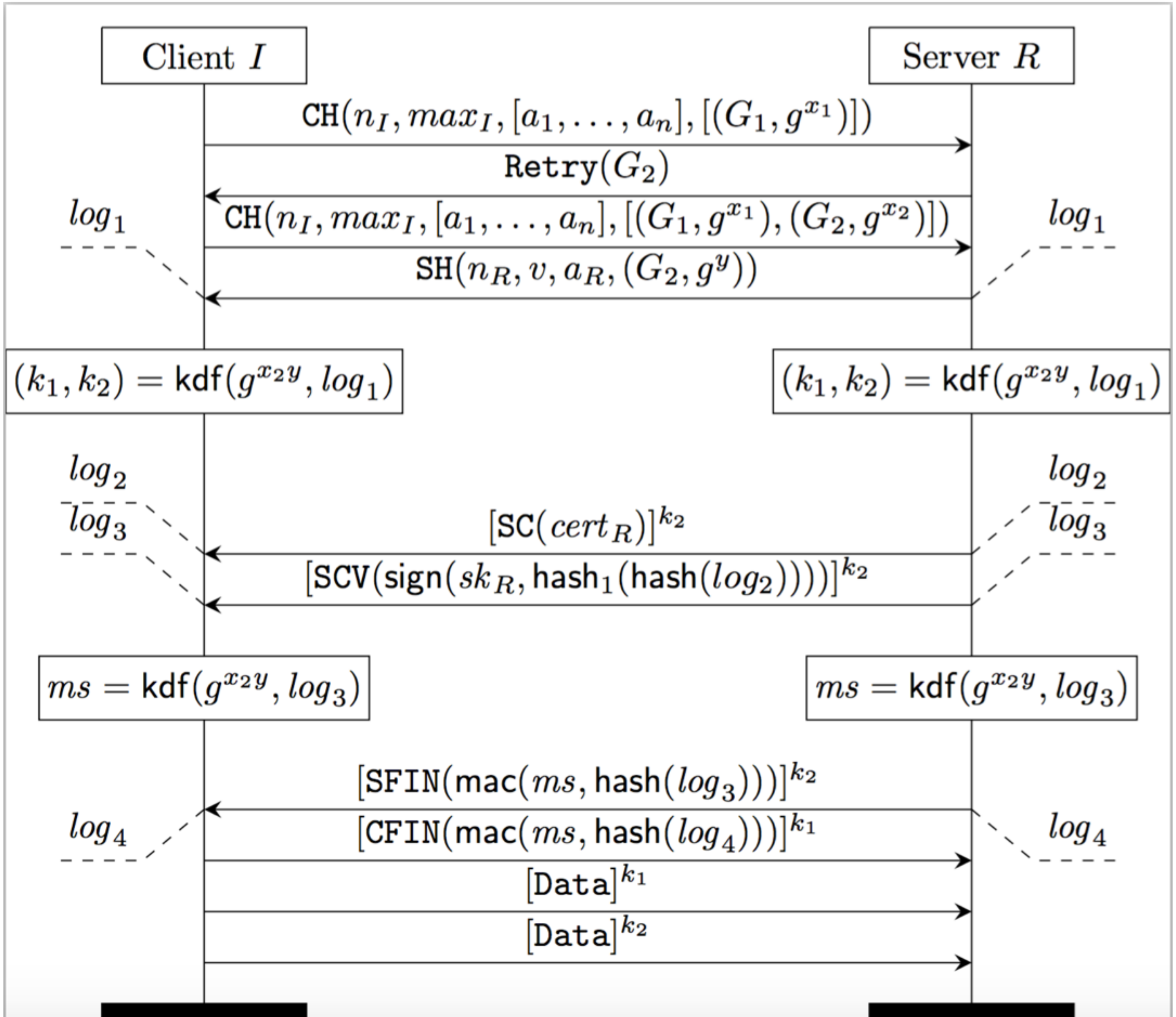
Faster: lower latency with 1 round-trip

- 0-round trip mode also available

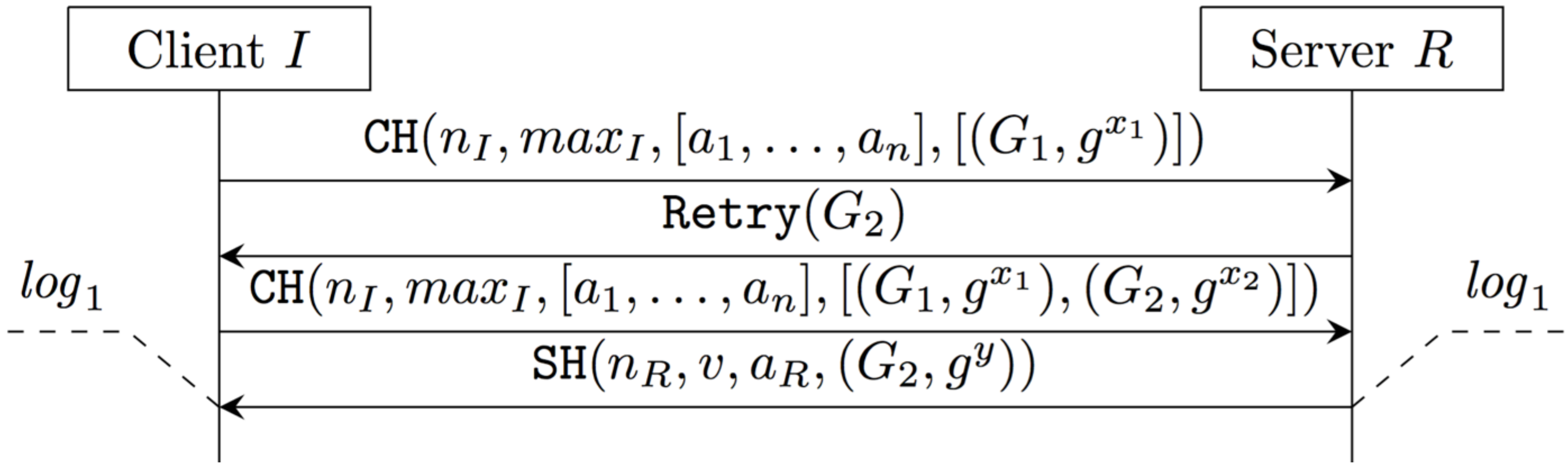
Crypto proofs built side-by-side with standardization

- Active participation by a large group of researchers
- Proofs in multiple symbolic and computational models
- Verified implementation in miTLS (ongoing work)

TLS 1.3 Negotiation Sub-Protocol

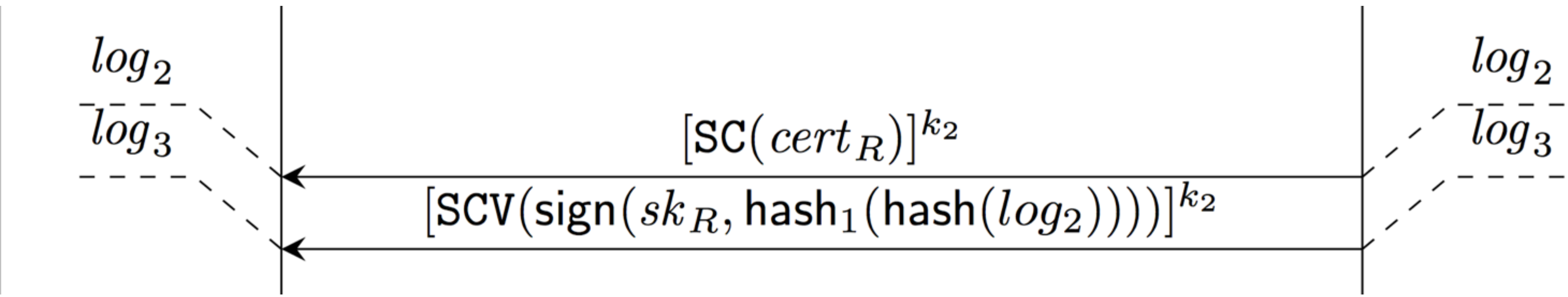


1: Group Negotiation with Retry



- Server can ask client to retry with another group
  - What if attacker sends a bogus Retry?
- ***Idea:*** The transcript hashes *both* hellos and retry to prevent tampering of Retry messages.

2: Full Transcript Signatures



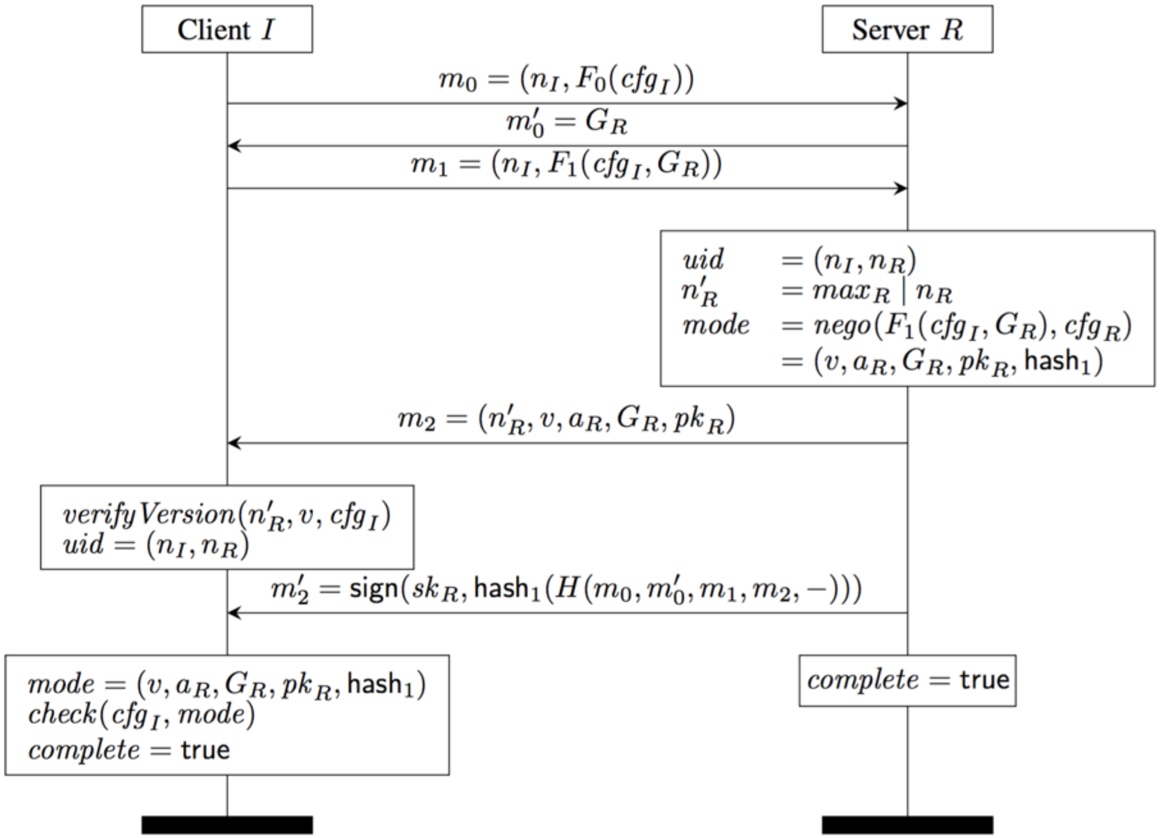
- Client and Server both sign *full* transcript
  - Only SHA-256 or newer hash algorithms allowed
  - Downgrade resilience can rely only on signatures
  - Logjam-like attacks are prevented!

3: Preventing Version Downgrade

- Clients and servers will support TLS 1.2 for a long time
  - TLS versions evolve slowly on the web: TLS 1.0 is still the most widely deployed version
- An attacker may downgrade TLS 1.3 to TLS 1.2 and then reuse known downgrade attacks!
  - TLS 1.3 clients and servers will still be vulnerable to Logjam
- ***Idea***: the server includes maximum supported version in server nonce (64 upper bits)
  - server nonce is signed in all versions TLS 1.0-1.3
  - only protects signature ciphersuites, not RSA encryption

TLS 1.3 is Downgrade Resilient

- We prove downgrade resilience for the *negotiation sub-protocol* of TLS 1.3 [S&P 2016]



Attacks on Legacy Crypto in TLS

• RC4	Keystream biases	[Mar'13]
• Lucky 13	MAC-Encode-Encrypt CBC	[May'13]
• POODLE	SSLv3 MAC-Encode-Encrypt	[Dec'14]
• FREAK	Export-grade 512-bit RSA	[Mar'15]
• LOGJAM	Export-grade 512-bit DH	[May'15]
• SLOTH	RSA-MD5 signatures	[Jan'16]
• DROWN	SSLv2 RSA-PKCS#1v1.5 Enc	[Mar'16]

- TLS was supposed to prevent downgrade attacks
- What went wrong? How do we fix it in TLS 1.3?

Final Thoughts

- Legacy crypto is strangely hard to get rid of, but we have to keep trying to kill broken primitives
- We need new downgrade resilient protocols

- In prior versions, TLS suffered a large time lag between standardization and proofs of security
- With TLS 1.3, researchers are closing this gap

- More details, papers, demos are at:

http://mitls.org