Secure multipath key exchange

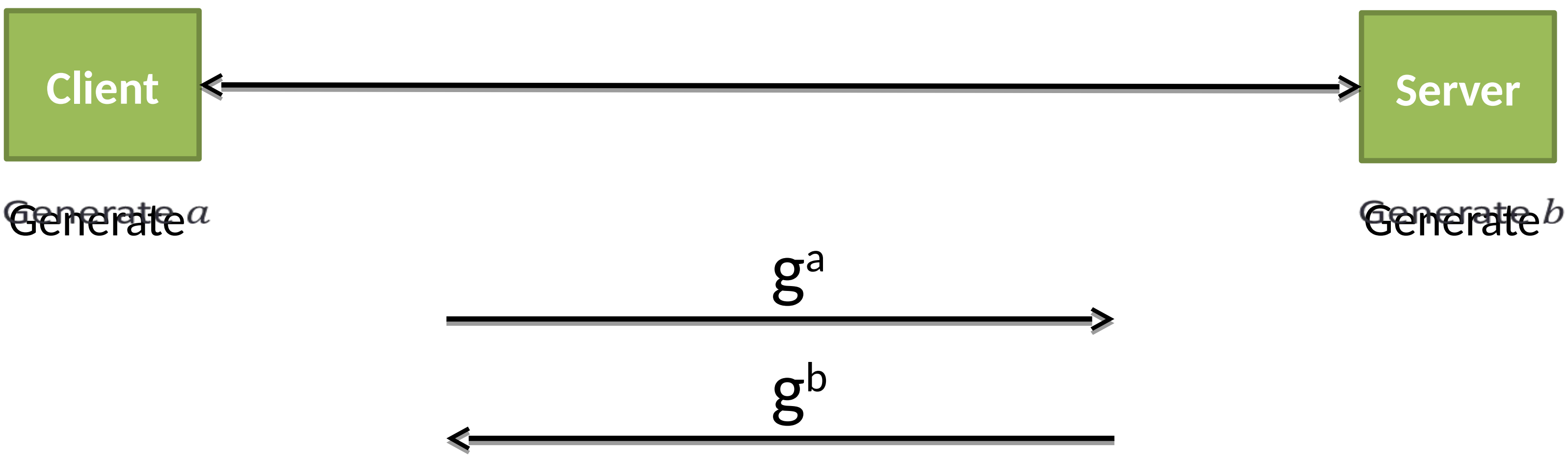
Sergiu Costea, Marios Choudary and Costin Raiciu

University Politehnica of Bucharest

SSICLOPS

Ubiquitous (Opportunistic) Encryption

- TCPCrypt based on Diffie-Hellman Key Exchange

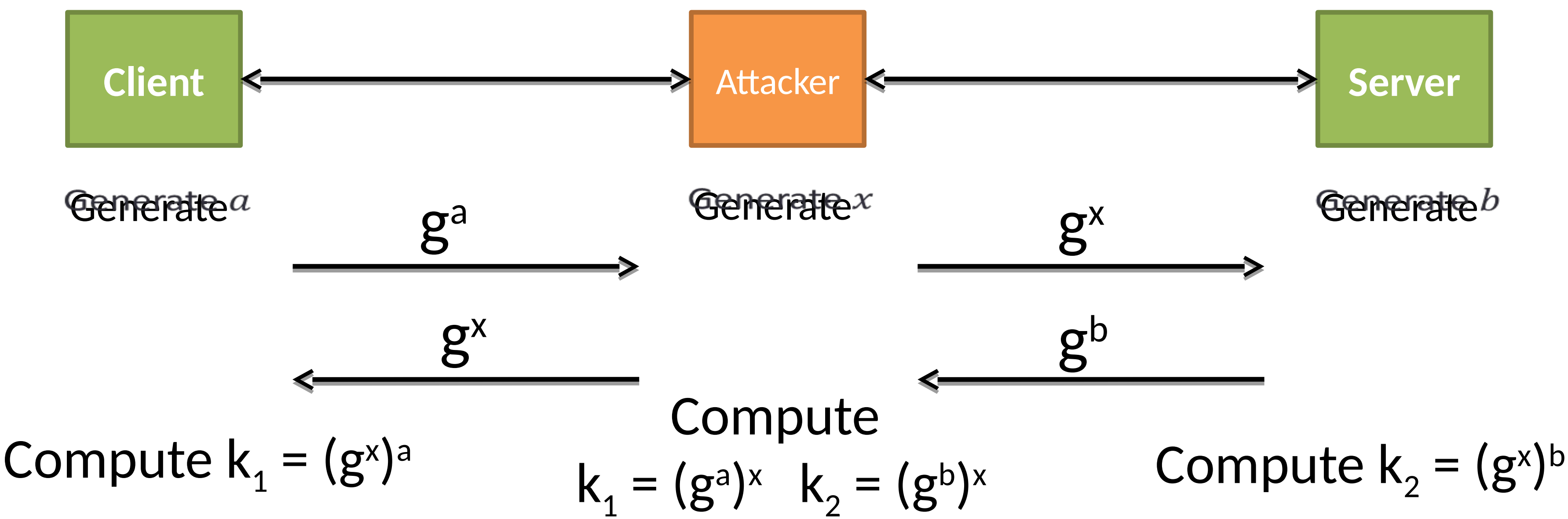


Compute $k = (g^b)^a$

Compute $k = (g^a)^b$

Ubiquitous (Opportunistic) Encryption

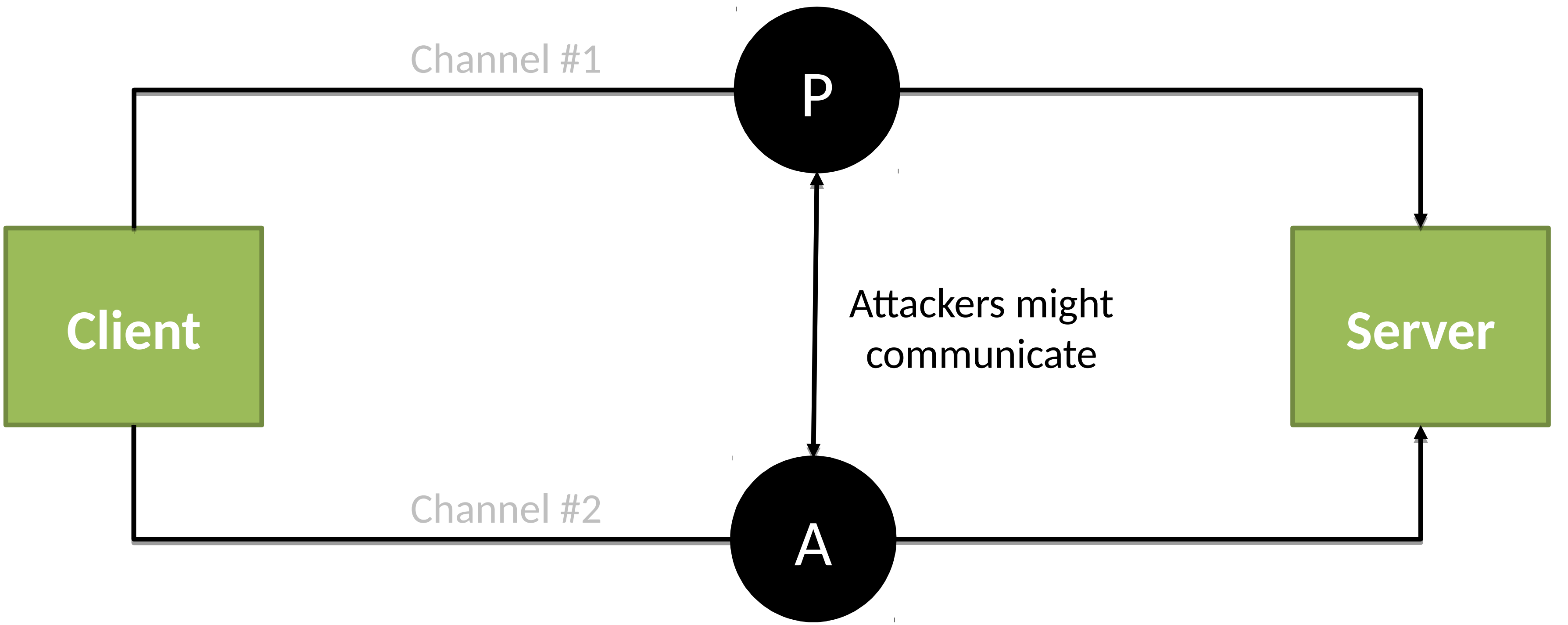
- Who am I exchanging keys with?



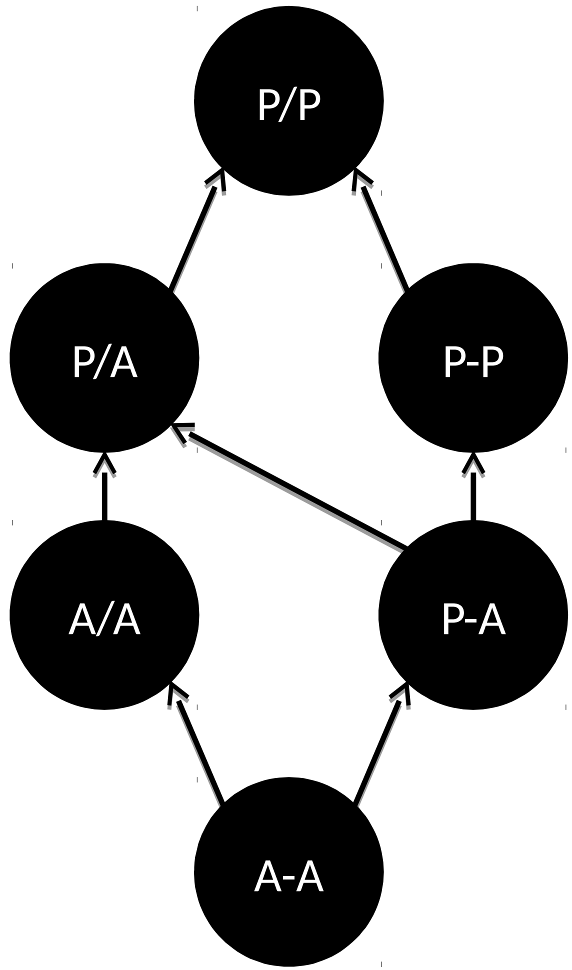
But networks are multipath

- Can we use multiple paths between endpoints to make attacks harder?

Types of attacks



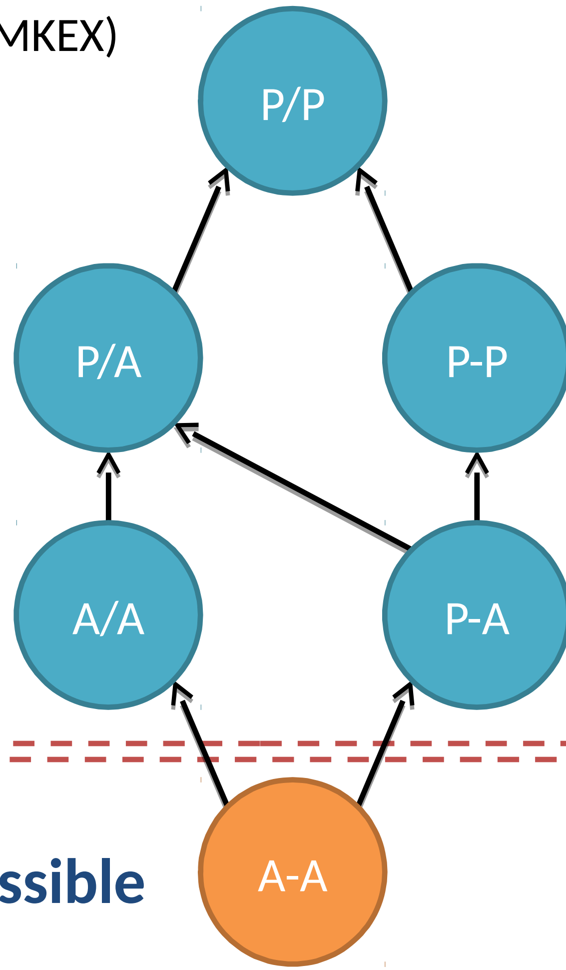
Threat Hierarchy



P	Passive
A	Active
-	Communication
/	No communication

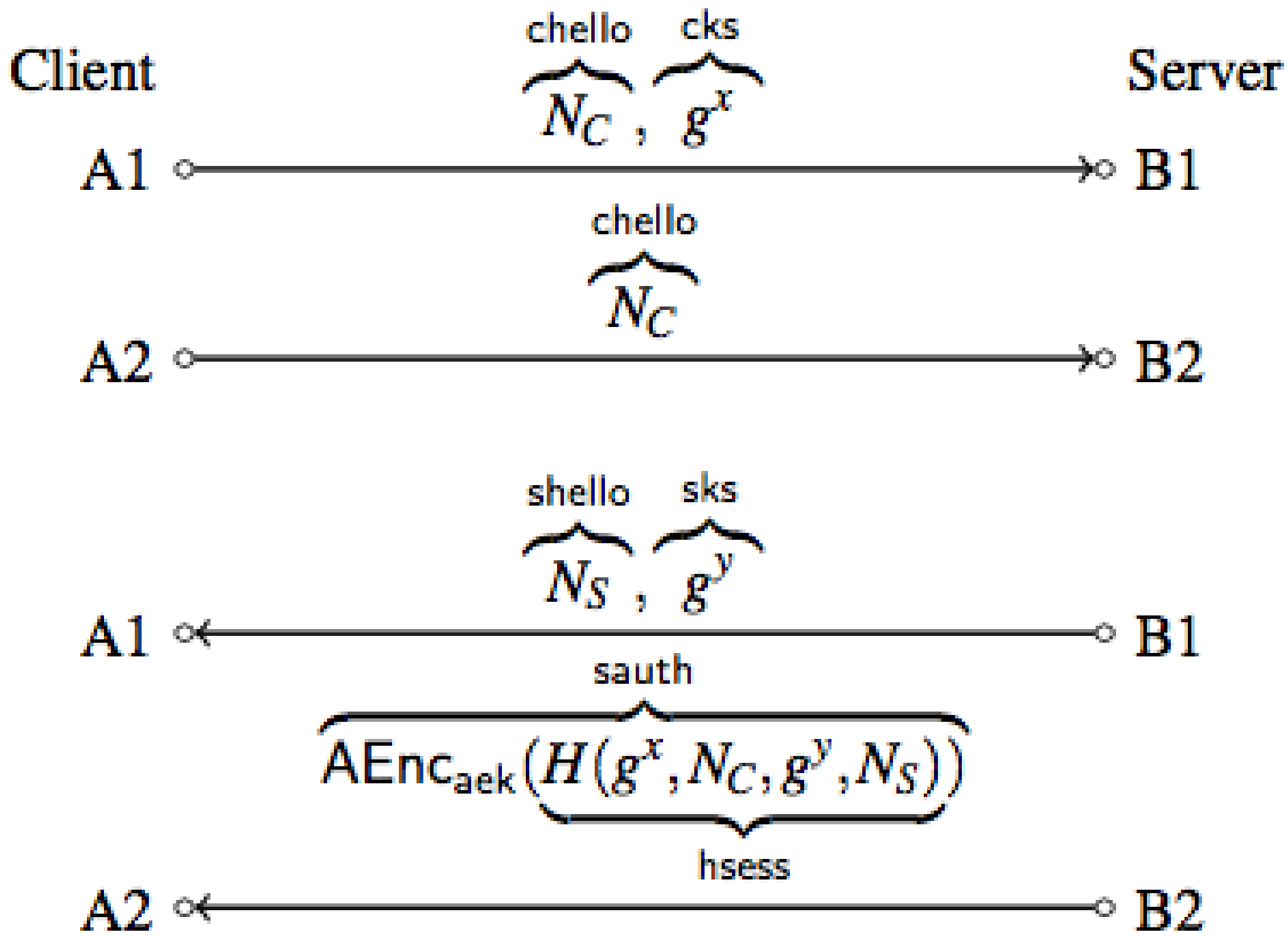
Threat Hierarchy

Secure Multipath Key Exchange Protocol (SMKEX) secure against



Impossible

SMKEX


Client
Server
chello
cks
N_C , g^x
A1
B1
chello
N_C
A2
B2
shello
sks
N_S , g^y
A1
B1
sauth
$\mathsf{AEnc}_{\mathsf{aek}}(H(g^x, N_C, g^y, N_S))$
hsess
A2
B2

MTLS = SMKEX + TLS

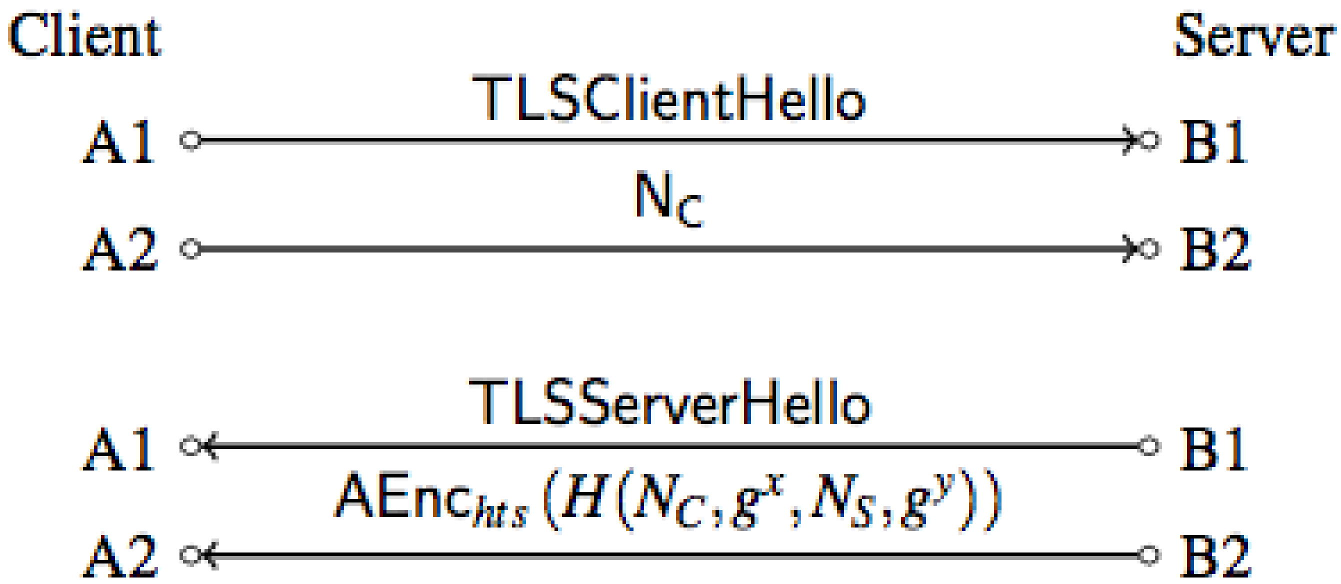


Figure 7: Multipath TLS protocol (*MTLS*). The first path executes the standard TLS key exchange, while the second path is used to validate keying information similarly to SMKEX.

SMKEX protection in a nutshell

Protocol	*P-A*, *A/A*		*A-A*	
	Auth	Rogue	Auth	Rogue
SMKEX	✓	-	X	-
TLS	✓	X	✓	X
MTLS	✓	✓	✓	X

Table 2: Comparison between the security features of SMKEX, TLS and MTLS

Implementation

- Library implementation of SMKEX over separate TCP connections
- Same cost and latency as single path DH

Evaluation: connection setup latency

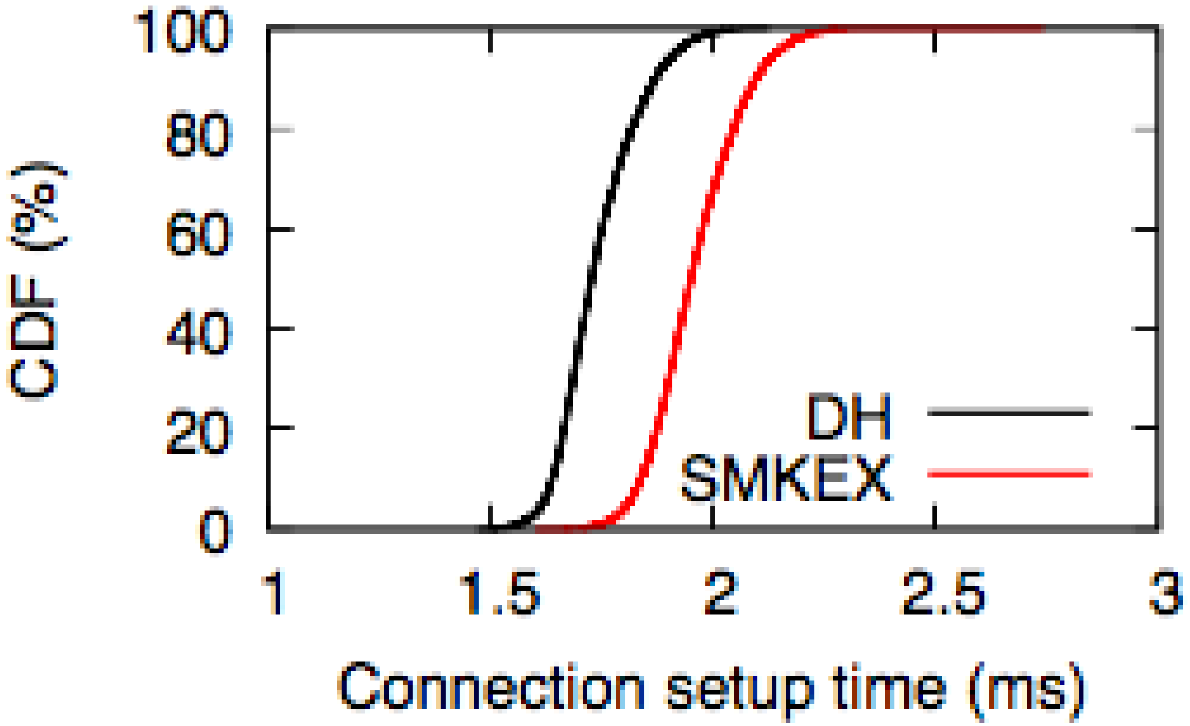
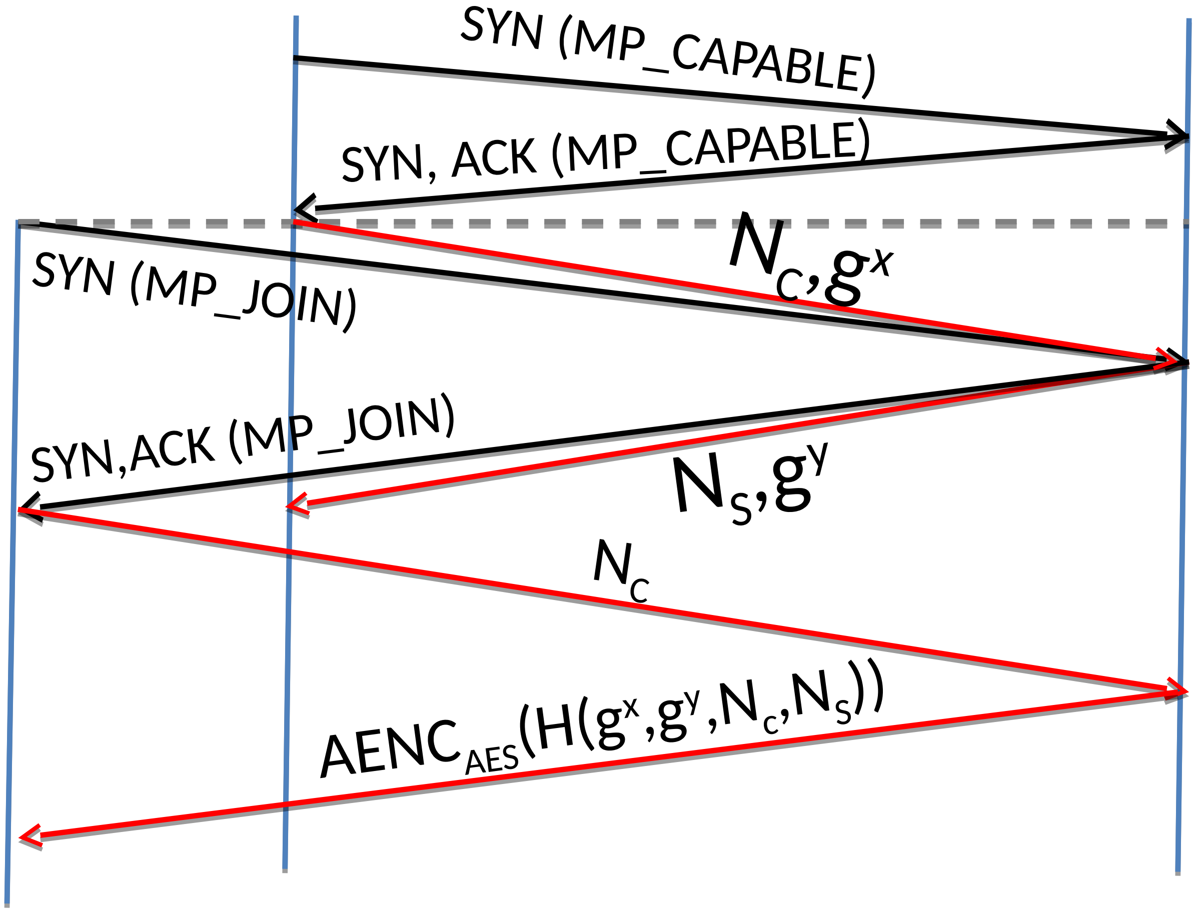
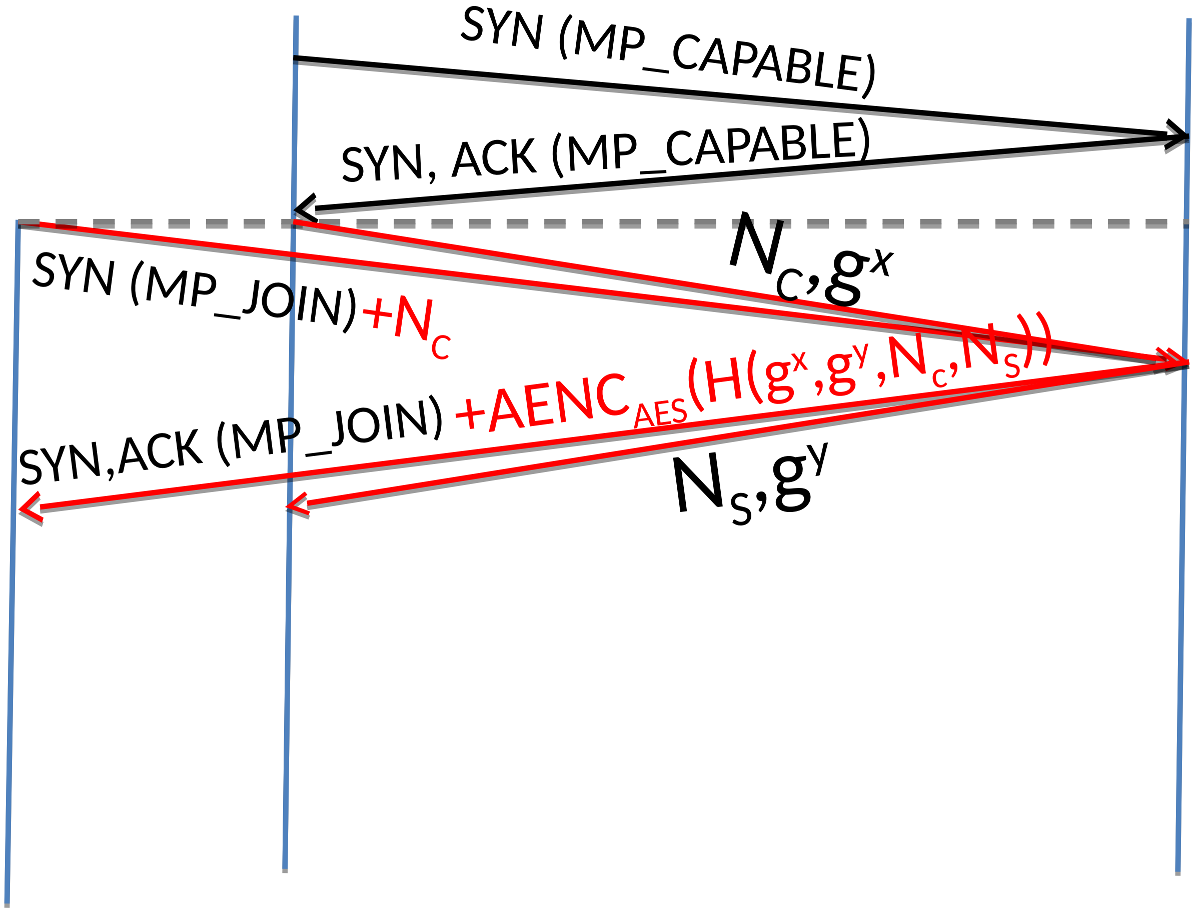


Figure 10: CDF of connection setup time, RTT=0.2ms

SMKEX over MPTCP today



SMKEX over MPTCP optimised



Ongoing work: MPTCP integration

- Added subflow preference API to MPTCP kernel
  - Subflow preference passed in one unused byte of the flags param of send / recv.
  - Scheduler that honors the preference
- Modified library implementation to use this code
- Now integrating the two parts.

Conclusions

- SMKEX is a step over DH/TLS

- Need to be able to send data on SYN_JOIN to reduce one RTT of crypto handshake