Secure multipath key exchange

Sergiu Costea, Marios Choudary and Costin Raiciu
University Politehnica of Bucharest
Ubiquitous (Opportunistic) Encryption

- TCPCrypt based on Diffie-Hellman Key Exchange

\[ \text{Client} \]
- Generate \( a \)
- Compute \( g^a \)
- Send \( g^a \) to \( \text{Server} \)

\[ \text{Server} \]
- Generate \( b \)
- Compute \( g^b \)
- Send \( g^b \) to \( \text{Client} \)

- Compute \( k = (g^b)^a \)  
- Compute \( k = (g^a)^b \)
Ubiquitous (Opportunistic) Encryption

- Who am I exchanging keys with?

Client

\[ g^a \rightarrow g^x \rightarrow g^b \]

Attacker

\[ k_1 = (g^a)^x \quad k_2 = (g^b)^x \]

Server

\[ k_1 = (g^x)^a \]

Compute \( k_2 = (g^x)^b \)
But networks are multipath

• Can we use multiple paths between endpoints to make attacks harder?
Types of attacks

Attackers might communicate.
Threat Hierarchy

P/P

P/A

P-P

P-A

A/A

A-A

P Passive
A Active
- Communication
/ No communication
Secure Multipath Key Exchange Protocol (SMKEX) secure against

- **P/P**
- **P/A**
- **P-P**
- **A/A**
- **P-A**
- **A-A**

Impossible
SMKEX

Client
A1

\(\overleftarrow{\text{chello}}\), \(\overleftarrow{\text{cks}}\)
\(N_C\), \(g^x\)

\(\overleftarrow{\text{chello}}\)
\(N_C\)

A2

Server
B1

\(\overrightarrow{\text{chello}}\), \(\overrightarrow{\text{cks}}\)
\(N_S\), \(g^y\)

\(\overrightarrow{\text{sauth}}\)

A1

\(\overrightarrow{\text{AEnc}_{\text{ae}}}(H(g^x, N_C, g^y, N_S))\)

\(\overrightarrow{\text{hsess}}\)

B1

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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>( P-A, A/A ) Auth</th>
<th>( P-A, A/A ) Rogue</th>
<th>( A-A ) Auth</th>
<th>( A-A ) Rogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMKEX</td>
<td>✓</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>TLS</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>MTLS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

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

SYN (MP_CAPABLE)

SYN, ACK (MP_CAPABLE)

SYN (MP_JOIN)

SYN, ACK (MP_JOIN)

$N_C, g^x$

$N_S, g^y$

$N_C$

$AENC_{AES}(H(g^x, g^y, N_C, N_S))$
SMKEX over MPTCP optimised

SYN (MP_CAPABLE)

SYN, ACK (MP_CAPABLE)

SYN (MP_JOIN) + NC

SYN, ACK (MP_JOIN) + AENC_{AES}(H(g^x, g^y, N_C, N_S))

NC, gx

NS, gy
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