Protokollsicherheit

Grundbegriffe der Kryptographie

Ziele

Einführung/Wiederholung grundlegender Konzepte der Kryptographie:

- Hash-Funktionen
- Message Authentication Codes (MACs)
- Symmetrische Verschlüsselung
- Asymmetrische Verschlüsselung
- Digitale Signaturen

Hash-Funktionen

Eine Hash-Funktion bildet eine Nachricht auf einen Hashwert fester Länge ab:

```
hash: bytes -> byte[k]
```

Anforderungen:

- `hash(m)` ist leicht zu berechenen
- Es ist schwer zu gegebenem `h` eine Nachricht `m` mit `hash(m)=h` zu finden.
- Es ist schwer eine Kollision zu finden, d.h. Nachrichten `m1` und `m2` mit `hash(m1)=hash(m2)`.

Anwendungen: Datenintegrität, Authentifizierung

Hash-Funktionen

Beispiel: A und B wollen das Spiel spielen, in dem jeder eine Zahl wählt und in dem der gewinnt, der die größere Zahl gewählt hat.

Hash-Funktionen können benutzt werden, um zu verhindern dass einer der Spieler darauf wartet, dass der andere seine Zahl sagt und dann mit einer größeren Zahl antwortet.

1. A und B wählen jeweils Zahlen n_A und n_B.
2. $A \to B : \texttt{hash}(n_A)$
3. $B \to A : \texttt{hash}(n_B)$
4. $A \to B : n_A$
5. $B \to A : n_B$

Hash-Funktionen

Beispiel: Secure Hash Algorithm (SHA-1)

- Entwickelt von NIST und NSA auf Basis von MD4.
- Erzeugt 160-Bit Hashwert.
- Verarbeitet Eingabe in 512-Bit-Blöcken.

  Besteht die Eingabe aus Blöcken $B_1 B_2 \dots B_n$, so wird eine Folge von 160-Bit Werten $H_1, \dots, H_{n+1}$ iterativ berechnet.
  - Initialisierung von H_1.
  - $H_{i+1} = f(H_i, B_i)$

  Der Hash H wird dann aus H_{n+1} gebildet ($H = g(H_{n+1})$).
- Algorithmen für f und g basieren auf DES.
- 2006 wurde in SHA-1 eine Schwäche in der Kollisionsresistenz entdeckt. $\Rightarrow$ ähnlichen SHA-2 benutzen (SHA-3 aktuell ausgeschrieben)

Message Authentication Codes

MACs (auch keyed hashes genannt) sind Codes fester Länge, welche Integrität und Authentizität einer Nachricht bezeugen.

```
mac : key * bytes -> bytes[k]
```

Statt der Nachricht `m` sendet man das Paar `(m, mac(k,m))`.

Anforderungen:

- `mac(k,m)` ist leicht zu berechnen.
- Ohne den Schlüssel `k` ist `mac(k,m)` für eine neue Nachricht `m` schwer zu erzeugen, selbst wenn man `(m1,mac(k,m1)),...,(mn,mac(k,mn))` kennt.

Message Authentication Codes

Wenn nur Sender und Empfänger den Schlüssel `k` kennen, nicht aber der Angreifer, so können sie Integrität und Autentizität der Nachricht überprüfen.

Integrität: Ein Angreifer kann die Nachricht `m` nicht ändern:

- Änderung zu `(m1,mac(k,m))` kann vom Empfänger bemerkt werden, indem er `mac(k,m1)` berechnet und mit dem empfangenen `mac(k,m)` vergleicht.
- Änderung zu `(m1,mac(k,m1))` ist wegen der Annahmen an `mac` schwer.

Authentizität: Ein Angreifer kann selbst keine gültigen Paare `(m,mac(k,m))` erzeugen. Er kann höchstens alte Nachrichten neu senden.

Message Authentication Codes

Implementierung durch Hash-Funktionen:

- Versuch 1

$$\texttt{mac}(k, m) = \texttt{hash}(k \| m)$$

  Problematisch bei blockbasierten Hash-Funktionen, da man dann dann in vielen Fällen $\texttt{mac}(k, m \| m')$ aus $\texttt{mac}(k, m)$ ableiten kann.

($m_1 \| m_2$ ist Konkatenierung von m_1 und m_2)

Message Authentication Codes

Implementierung durch Hash-Funktionen:

- Versuch 1

  $$\texttt{mac}(k, m) = \texttt{hash}(k \| m)$$

  Problematisch bei blockbasierten Hash-Funktionen, da man dann dann in vielen Fällen $\texttt{mac}(k, m \| m')$ aus $\texttt{mac}(k, m)$ ableiten kann.

- Versuch 2

  $$\texttt{mac}(k, m) = \texttt{hash}(m \| k)$$

  Kennt man bereits m_1 und m_2 mit $\texttt{hash}(m_1)$ und $\texttt{hash}(m_2)$, so kann man dann MACs für diese Nachrichten erzeugen, ohne den Schlüssel zu kennen.

($m_1 \| m_2$ ist Konkatenierung von m_1 und m_2)

Message Authentication Codes

Implementierung durch Hash-Funktionen:

- RFC 2104

$$\texttt{mac}(k, m) = \texttt{hash}(k \oplus opad \parallel \texttt{hash}(k \oplus ipad \parallel m))$$

wobei $ipad = \texttt{0x363636...36}$ und $opad = \texttt{0x5C5C5C...5C}$.

Symmetrische Verschlüsselung

```
keygen: unit -> key
encrypt: key * bytes -> bytes
decrypt: key * bytes -> bytes
```

```
decrypt(k, encrypt(k, m)) = m
```

Anforderung:

- Ohne `k` ist es schwer `m` aus `encrypt(k, m)` zu berechnen.

Beispiele: DES, AES, ...

Beispielprotokoll: Otway-Rees [1987]

Ziel: A und B wollen sich auf einen Sitzungsschlüssel K_{AB} einigen. Der Schlüssel soll von S erzeugt werden und sonst niemandem bekannt werden.
A und B haben bereits Langzeitschlüssel zur Kommunikation mit S.

1. $A \to B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \to S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \to B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \to A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Asymmetrische Verschlüsselung

```
keygen: unit -> key * key
encrypt: key * bytes -> bytes
decrypt: key * bytes -> bytes
```

`keygen()` erzeugt ein Schlüsselpaar `(pk,sk)` aus einem öffentlichen und einem geheimen Schlüssel.

Nachrichten werden mit dem öffentlichen Schlüssel verschlüsselt. Der Empfänger kann sie mit seinem privaten Schlüssel entschlüsseln.

```
decrypt(sk, encrypt(pk, m)) = m
```

Asymmetrische Verschlüsselung

```
keygen: unit -> key * key
encrypt: key * bytes -> bytes
decrypt: key * bytes -> bytes
```

Anforderungen

- Sei `(pk,sk)` ein Schlüsselpaar. Dann ist es schwer von `pk` auf `sk` zu schließen.
- Es ist schwer aus `encrypt(pk,m)` und `pk` allein die Nachricht `m` zu berechnen.

Beispiele: RSA, Elgamal, DSA

Elgamal Verschlüsselung

Schlüsselerzeugung:

- Erzeuge endliche zyklische Gruppe $(G, \cdot, 1, (-)^{-1})$ mit Ordnung p und erzeugendes Element g. (D.h. $G = \{g^k \mid k \in \mathbb{N}\}$ und $|G| = p$)
  Beispiel: $\mathbb{Z}_{p+1}^* = (\{1, \ldots, p\}, \cdot, 1)$ für große Primzahl $p + 1$.
- Wähle Zufallszahl $x \in \{0, \ldots, p - 1\}$.
- Privater Schlüssel: x
- Öffentlicher Schlüssel: (G, g, g^x)
  (Gruppe der Form $\mathbb{Z}_p^*$ kann durch p beschrieben werden.)

Elgamal Verschlüsselung

Verschlüsselung:
Sei (G, g, h) der öffentliche Schlüssel von A.
Die zu verschlüsselnde Nachrichten sei als Element m von G gegeben.

- Wähle eine Zufallszahl $k \in \{0, \ldots, |G| - 1\}$.
- Die verschlüsselte Nachricht ist $(g^k, m \cdot h^k)$.

Entschlüsselung:
Sei (s_1, s_2) die verschlüsselte Nachricht.
Sei x der zu (G, g, h) passende private Schlüssel, d.h. $h = g^x$.

- Berechne $s_1^{-x} \cdot s_2$.
  Es gilt
  $s_1^{-x} \cdot s_2 = (g^k)^{-x} \cdot m \cdot h^k = (g^k)^{-x} \cdot m \cdot (g^x)^k = g^{-kx} \cdot m \cdot g^{xk} = m$,
  d.h. die Nachricht ist damit entschlüsselt.

Beispielprotokoll: Needham-Schroeder [1978]

Protokoll zur gegenseitigen Authentisierung von A und B und zur Etablierung eines gemeinsamen Geheimnis (N_A, N_B).

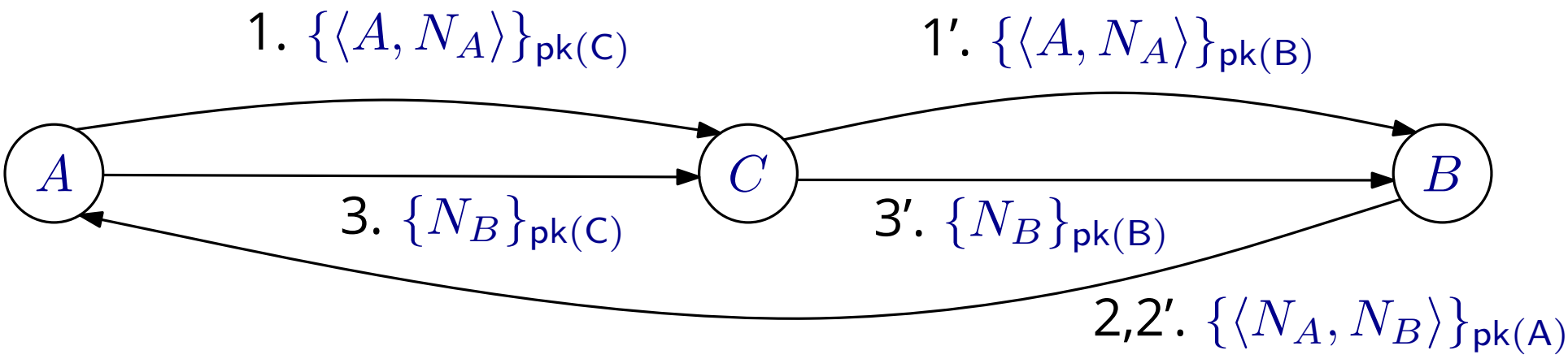
1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle N_A, N_B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Beispielprotokoll: Needham-Schroeder [1978]

Protokoll zur gegenseitigen Authentisierung von A und B und zur Etablierung eines gemeinsamen Geheimnis (N_A, N_B).

1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle N_A, N_B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Angriff [Lowe 1996]

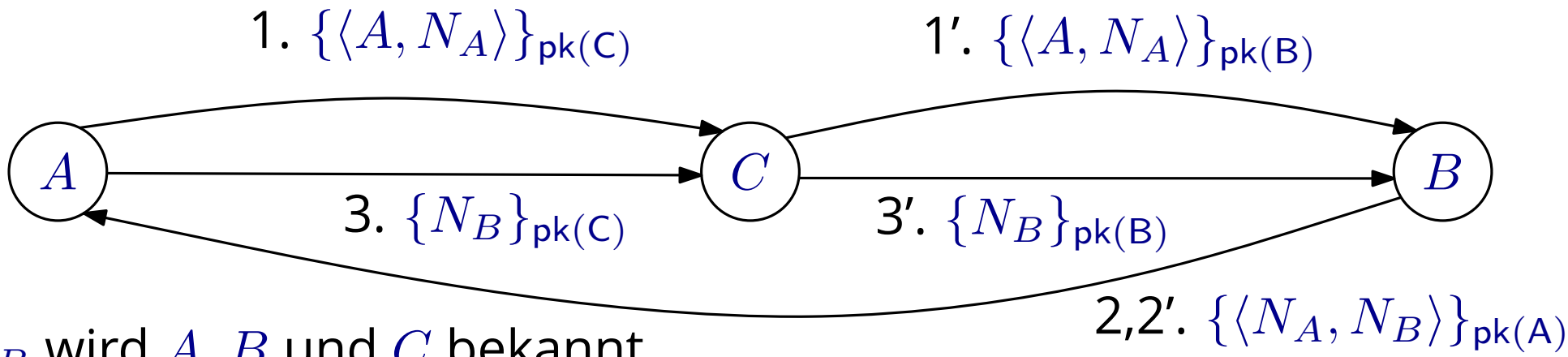


Beispielprotokoll: Needham-Schroeder [1978]

Protokoll zur gegenseitigen Authentisierung von A und B und zur Etablierung eines gemeinsamen Geheimnis (N_A, N_B).

1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle N_A, N_B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Angriff [Lowe 1996]



N_B wird A, B und C bekannt.
A glaubt mit C zu reden (Lauf 1,2,3).
B glaubt mit A zu reden (Lauf 1′,2′,3′).

Beispielprotokoll: Needham-Schroeder [1978]

Protokoll zur gegenseitigen Authentisierung von A und B und zur Etablierung eines gemeinsamen Geheimnis (N_A, N_B).

1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle N_A, N_B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Lowes korrigiertes Protokoll [1996]:

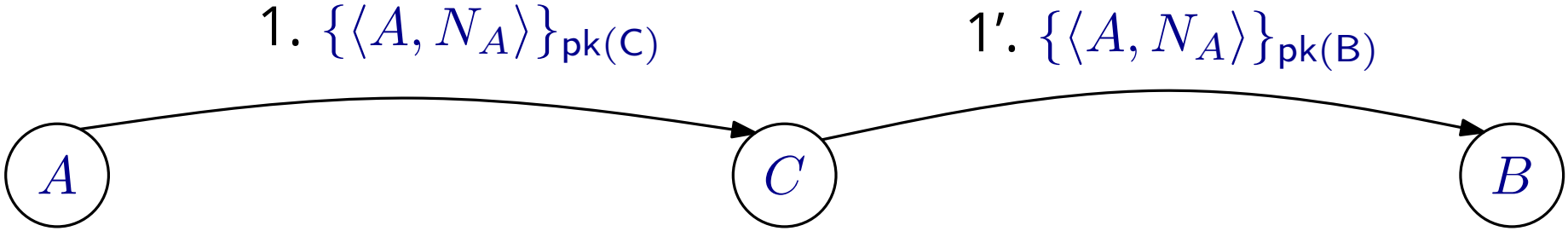
1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle \langle N_A, N_B \rangle, B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Die Sicherheit des korrigierten Protokolls wurde in mehreren Modellen bewiesen, z.B. im symbolischen Modell mit Proverif.

Needham-Schroeder-Lowe Protokoll mit Elgamal

1. $A \to B$: $\{\langle A, N_A \rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle \langle N_A, N_B \rangle, B \rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

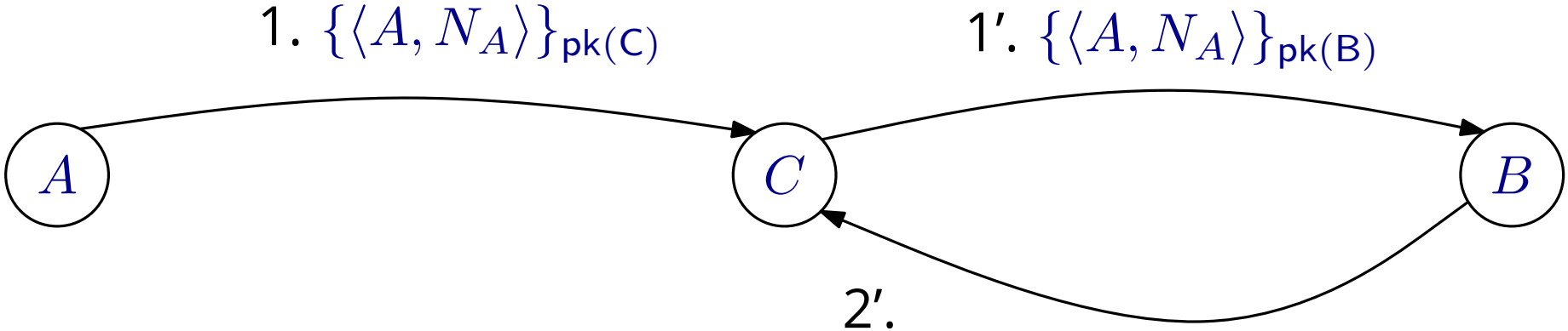
Angriff:



Needham-Schroeder-Lowe Protokoll mit Elgamal

1. $A \to B$: $\{\langle A, N_A\rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Angriff:



$\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk(A)}}$

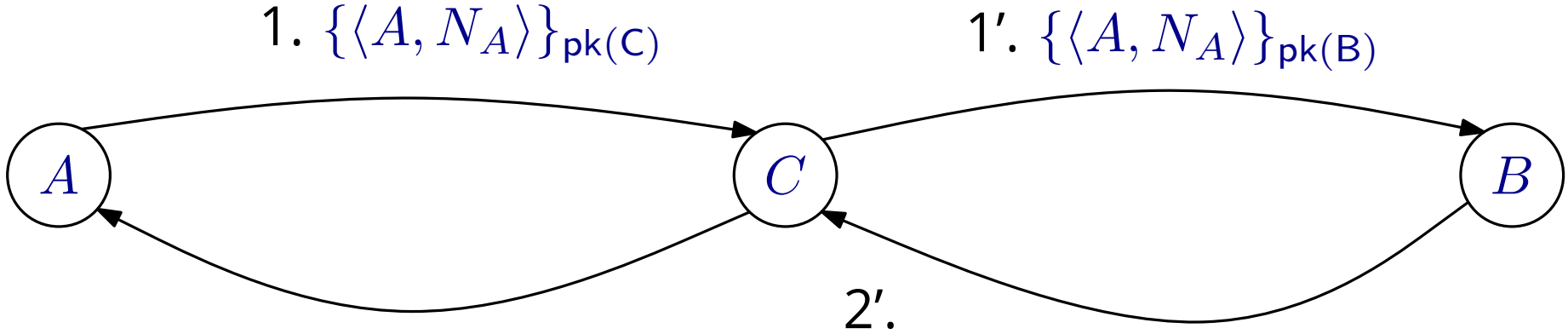
$= (g^k, \mathsf{pk}(A)^k \cdot \langle\langle N_A, N_B\rangle, B\rangle)$

$= (g^k, \mathsf{pk}(A)^k \cdot g^{N'_A + 2^L \cdot N'_B + 2^{2^L} \cdot B'})$

Needham-Schroeder-Lowe Protokoll mit Elgamal

1. $A \to B$: $\{\langle A, N_A\rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Angriff:



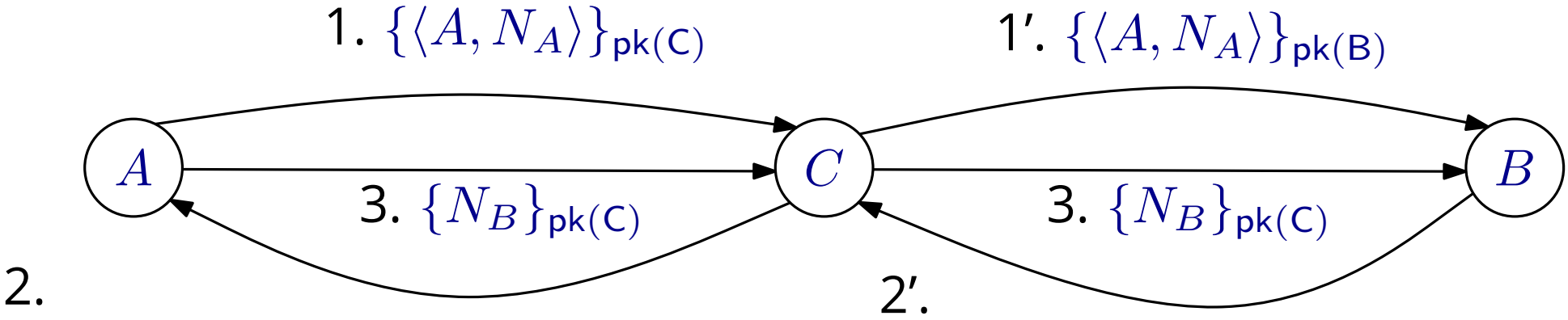
$\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk(A)}} \cdot g^{2^{2^L}\cdot C' - 2^{2^L}\cdot B'}$
$= \{\langle\langle N_A, N_B\rangle, C\rangle\}_{\mathsf{pk(A)}}$

$\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk(A)}}$
$= (g^k, \mathsf{pk}(A)^k \cdot \langle\langle N_A, N_B\rangle, B\rangle)$
$= (g^k, \mathsf{pk}(A)^k \cdot g^{N'_A + 2^L \cdot N'_B + 2^{2^L}\cdot B'})$

Needham-Schroeder-Lowe Protokoll mit Elgamal

1. $A \to B$: $\{\langle A, N_A\rangle\}_{\mathsf{pk}(B)}$
2. $B \to A$: $\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk}(A)}$
3. $A \to B$: $\{N_B\}_{\mathsf{pk}(B)}$

Angriff:



$\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk(A)}} \cdot g^{2^{2^L}\cdot C' - 2^{2^L}\cdot B'}$
$= \{\langle\langle N_A, N_B\rangle, C\rangle\}_{\mathsf{pk(A)}}$

$\{\langle\langle N_A, N_B\rangle, B\rangle\}_{\mathsf{pk(A)}}$
$= (g^k, \mathsf{pk}(A)^k \cdot \langle\langle N_A, N_B\rangle, B\rangle)$
$= (g^k, \mathsf{pk}(A)^k \cdot g^{N'_A + 2^L \cdot N'_B + 2^{2^L}\cdot B'})$

Needham-Schroeder-Lowe Protokoll mit Elgamal

Problem: Elgamal-Verschlüsselung erlaubt es, verschlüsselte Nachrichten zu verändern, ohne sie zu entschlüsseln.

Ist (s_1, s_2) der Code für $m \in G$, so ist $(s_1, m' \cdot s_2)$ der Code für $m \cdot m' \in G$.

Digitale Signaturen

Signaturen dienen wie MACs dazu, die Authentizität von Nachrichten sicherzustellen.

```
sign : key * bytes -> bytes[k]
verify : key * bytes * bytes[k] -> bool
```

Zum Signieren benutzt man einen privaten Schlüssel (`sign(sk,m)`), zum Überprüfen den zugehörigen öffentlichen (`verify(pk,m,sign(sk,m))=true`).

Anforderungen:

- Ohne den Schlüssel `sk` ist `sign(sk,m)` für eine neue Nachricht `m` schwer zu erzeugen, selbst wenn man alte `(m1,sign(sk,m1)),...,(mn,sign(sk,mn))` kennt.
- Unleugbarkeit: Gilt `verify(pk,m,s) = true`, so hat `s` mit hoher Wahrscheinlichkeit die Form `sign(sk,m)`.

Digitale Signaturen

Digitale Signaturen können mit Hashfunktionen und asymmetrischen Verschlüsselungsverfahren implementiert werden.

```
sign(sk, m) = encrypt(sk, hash(m))
```

```
verify(pk, m, s) = (hash(m) = decrypt(pk, s))
```