1 Protocols problem set

This document outlines several protocols that we recommend for the crypto ladder. The proposed protocols are listed in roughly increasing order of proof difficulty within the crypto world:

- Basic-Hash (Example 1), a RFID tag/reader access protocol, meant to provide unlinkability, a strong notion of anonymity, between the tags. This is the base pattern for electronic passport authentication.
- Signed DH (Example 2), a simple key-exchange providing secrecy and authentication, which is the historic pattern for SSH/TLS.
- Signed KEM (Example 3), a bilateral authenticated variant relying on KEMs, illustrating subtle attacks against key-exchanges.
- Signed DH+KEM (Example 4), a hybrid variant with both KEM and DH following the pattern for the ongoing hybridization of SSH/TLS.
- A simplified NTOR protocol (Example 5), a DH key-exchange using long term DH keys for authentication instead of signatures, which is the base pattern for Wireguard and PQXDH.

The rational behind this set:

- provide a variety of security properties (secrecy, authentication, privacy);
- have examples linked with widely deployed real-world applications or real-world attacks;
- start with a very simple example and increase difficulty.

Notations In this document, secret keys and key pairs are denoted by using the subscripts sk/pk to distinguish between secret/public parts. We rely on usual algorithms for KEM, signatures and DH. We use a generic hash function Hash without a fixed arity, modelers may use the most canonical way to represent those computations in their tools.

1.1 Modeling protocols

Protocols are asynchronous programs executed between several parties over an often untrusted network. Due to the variety of possible modeling choices for both execution models and security properties, we do not commit in this presentation to a single particular one. Each problem will be specified at a high-level by giving a message sequence chart of an expected run between two (or more) honest agents. An agent can be characterized by a role and some long term material. Typically, for a key-exchange setting we can denote by $S(s_{sk}, s_{pk})$ a server owning the signing key s_{sk} and $C(s_{pk})$ a client that already knows the corresponding public and thus wants to talk to this particular server.

Each problem may be "solved" in a variety of way based on many different modeling choices. We detail those choices below.

Execution scenario The simplest execution scenario for each protocol is where there a single party for each possible role, and each party only participates in a single session. However, we want to consider scenarios closer to real world executions, with an unbounded number of sessions and possible parties. We thus have the following options:

- **bounded vs unbounded number of sessions** - Scenarios with a bounded number of sessions corresponds to some $C(s_{pk})$ interacting at most a bounded fixed number of times with the corresponding $S(s_{sk}, s_{pk})$. However, we may want to allow this C and S to execute between themselves the protocol an unbounded number of time, which corresponds to an unbounded number of sessions settings.
- **bounded vs unbounded number of agents** - In basic scenarios, we may only consider a bounded number of possible identities, and typically consider that there is a single server. But we may also want to consider the case where there are for each role an unbounded number of possible agents/identities. This means that there typically would be a set of possible public keys $\{s^i_{pk}\}$ for all the possible identities.

Threat model We consider that the attacker is at the level of the network communications. Its power may vary depending on the following

- **passive vs active attacker** - a passive attacker can intercept and read all messages sent over the network, but it cannot tamper with them. An active attacker in addition modify at will any message sent over the network and add its own.
- **compromise of ephemeral and/or long term material** - the attacker might be able to target some users, and partially or fully corrupt them. In the basic scenario, the attacker cannot do any such compromise. It may then be able to corrupt the long term material of an agent, typically the s_{sk} of a server (not that in the previous key exchange scenario, C did not have any long term secret material). The attacker might also be able to compromise the ephemeral material of a particular sessions of an agent, essentially leaking the ephemeral secret keys of this session.

Modeling of primitives For protocol analysis, primitives are most of the time abstracted away as abstract function calls.

- **computational vs symbolic approach** - in the computational approach, hardness assumptions are made over the primitives, like IND-CCA for encryption, and the attacker is in effect restricted by what it cannot do w.r.t. to the primitives. In the symbolic approach, primitives are idealized, and typically, an encryption perfectly hides its content and we get nothing out of it unless we know the corresponding secret keys [1].

[1] In more details, equations define their only possible behaviors

- **primitive characterization** - each primitive might be modeled in a variety of way, with weaker or stronger assumptions over it. In the computational approach, we may model the encryption as IND-CPA or IND-CCA1 or IND-CCA2. The symbolic approach may consider a signature as perfect, as leaking the signed message, as suffering from DEO attacks and so on.

Security properties Finally, security properties can themselves be expressed in a variety of way, without always a clear consensus in the community. Rather than commit to a specific notion, we detail below some classes of attacks that are relevant in our setting.

Some very generic properties are:

- **Replay Attacks** - a message sent by a party can be replayed successfully several times to another party. (note that this is only meaningful in a multiple session scenario)
- **Data agreement** - several parties are expected to agree on a set of data at the end of an execution.

In the problem set, we will have a particular focus on key-exchanges, which have a variety of properties/attacks:

- **Authentication** - whenever a party expect to be talking to a given agent, it is indeed the case. Authentication might be **unilateral** (at the end of the key-exchange the client knows it is talking to the server owning s_{pk} but the server has no guarantee about the client), or **bilateral** (the client also owns some long term key, and the server is sure of which client it is talking to). In addition, authentication might be **injective or non-injective**, where it is injective if for each session of a client believing to be talking with some server, there is a single and distinct corresponding session on the server. If the authentication is non-injective, a single server session may validate authentication for several sessions of the client, which can typically happen in case of replay attacks.
- **Secrecy** - the key derived at the end of a session between two uncompromised participants should be secret. A strengthen variant is **Forward Secrecy**, where even if at some point the long term material of an agent is compromised, this does not affect the secrecy of all the previously completed exchanges.
- **Session independence** - for a given agent, compromising the ephemeral material of all sessions but one should not affect the security of the uncompromised session.
- **Unkown Key-Share attacks** - this class of attacks occurs when two honest agent might derive the same key, but in effect did not expect to be talking together, this typically may occur with a client C willing to talk to

a compromised server A, and the attacker makes it so that another server S derives the same key as C.

- **Key-Compromised impersonation** - in a scenario with a bilateral authenticated key exchange, when we compromise the long-term material of a given agent C, the attacker can of course impersonate C to some honest server S. However, it should not be the case that for later honest sessions of C that may run with unconpromised ephemeral material, an attacker can impersonate S. A classical scenario where this occurs is when two parties share a symmetric pre-shared (PSK) key used for authentication, leaking this PSK on the side of C allows to both impersonate C to S and S to C.

1.2 Problem solutions

We invite tool developers and users to contribute to this project. For each tool we thus encourage developpers to have a dedicated public repository containing in whatever format they wish, some full or partial solutions, along with tutorials or any other desired material.

In addition, we hope to aggregate in a repository (`https://github.com/charlie-j/fm-crypto-lib`) the full solutions file for each problem and tool, to enable for a quick browsing of the files, as well as provide for each problem a table summarizing all the existing contributions and giving some comparison points. To be able to build such a table, we ask that each solution for a particular problem comes with a standardized description summarizing the features of the model and analysis. See the README of the repository for the template description, as well as the tables.

2 Protocol problems

2.1 Basic-Hash

The protocol is given in Figure 1

Description For a set TAGS of valid identities, a RFID reader has a database of valid secret keys $\{t_{sk}\}_{T \in \text{TAGS}}$. A RFID tag with identity T and key t_{sk} authenticates to the reader by sampling a fresh challenge n, and sending the pair $(n, \text{Hash}(n, t_{sk}))$ to the reader, which looks into its database to see if the hash can be mapped to a valid identity and then answers true or false.

Properties This protocol should provide two guarantees against an active attacker:

- non-injective authentication - if the reader accepts some message as coming from some tag, the corresponding tag did send this message at some point in the past.

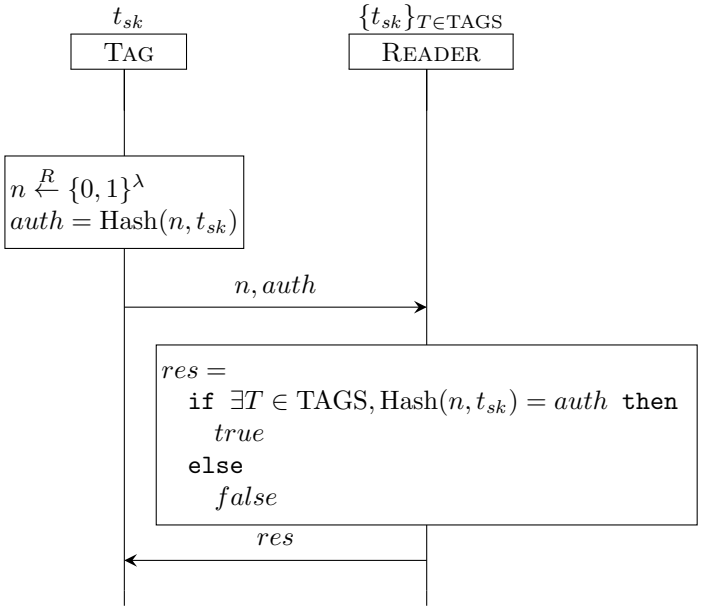

Figure 1: Basic Hash

- unlinkability - it should be impossible to decide whether two sessions of the protocol correspond to the same tag or not. That is, it should be impossible to distinguish a scenario where there is a single tag executing n times the protocol in sequence with a scenario where n distinct tags all execute the protocol each a single time.

Bonus: Show that injective authentication does not hold due to a replay attack. The corresponding attack trace is given in Appendix A, Figure 6.

Not that enabling compromise of material for this problem is not particularly relevant, as no particular security properties are preserved.

2.2 signed DH key exchanges

The protocol is given in Figure 2.

Description The server has a long term signature keypair s_{sk}, s_{pk}. This key is used to authenticate the server, while the client and the server exchange fresh DH values to derive a shared secret key.

Properties This protocol should provide two guarantees against an active machine-in-the-middle attacker:

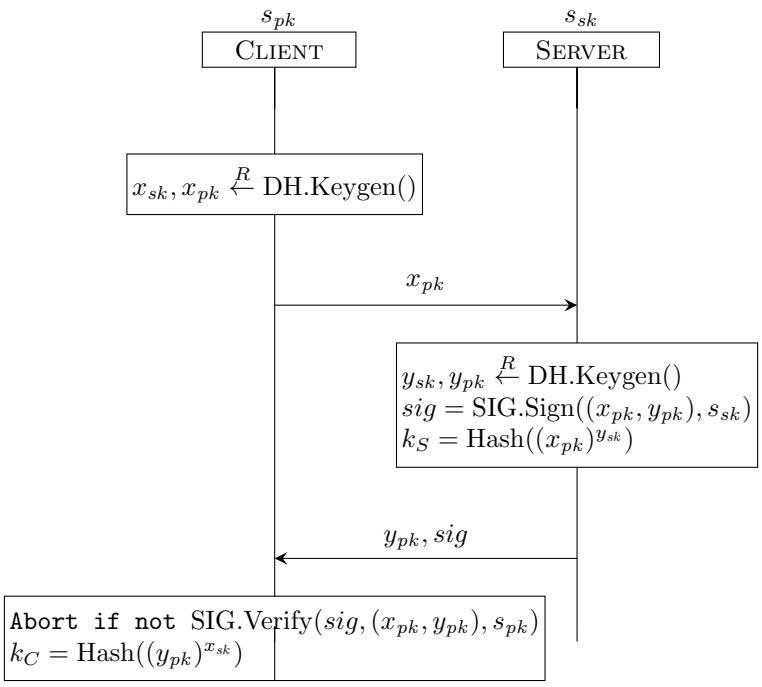

Figure 2: signed Diffie-Hellman key exchange

- injective unilateral client-side authentication - if a client derives a key with some parameters (the set of public keys), a corresponding server session derived the same key with the same parameters.
- secrecy - nobody except the two matching sessions can derive the key. The key might be proven to be indistinguishable from a random value.

2.3 signed KEM key exchanges

The protocol is given in Figure 3.

Description The client has a long term signature keypair c_{sk}, c_{pk}. This key is used to authenticate the client, while the client and the server exchange fresh KEM values to derive a shared secret key.

Properties This protocol should provide two guarantees against an active machine-in-the-middle attacker:

- bilateral authentication - TODO

- secrecy - nobody except the two matching sessions can derive the key. The key might be proven to be indistinguishable from a random value.

We want to consider in this problem a KEM that is just IND-CCA, the KEM may thus be implemented using an asymmetric encryption function KEM.AEnc, and with

$$\text{KEM.Encaps}(x_{pk}) := ss \xleftarrow{R} \{0,1\}^{\lambda}; ct \xleftarrow{R} \text{KEM.AEnc}(ss, x_{pk}); \text{return } (ct, ss)$$

Shows that when instantiating the KEM with an asymmetric encryption of a fresh secret, this property does not hold.

Bonus: when considering a scenario where some clients are willing to talk to dishonest parties, this protocol suffers from an Unknown Key Share attack. The corresponding attack trace is given in Appendix A, Figure 7.

Bonus: when allowing attackers to compromise ephemeral keys x_{sk}, we should on a good protocol still be able to prove that whenever a client session and a server session derive the same secret key, then if the corresponding client session x_{sk} was not compromised, the key is fully secret. This corresponds to session independence, where allowing to compromise the material of other sessions does not impact the security of uncompromised sessions. The corresponding attack trace is given in Appendix A, Figure 8.

2.4 signed DH+KEM key exchanges

The protocol is given in Figure 4.

Description The server has a long term signature keypair s_{sk}, s_{pk}. This key is used to authenticate the server, while the client and the server exchange fresh DH and KEM values to derive a shared secret key.

Properties This protocol should provide two guarantees against an active machine-in-the-middle attacker:

- authentication - if a client derives a key with some parameters (the set of public keys), a corresponding server session derived the same key with the same parameters.
- secrecy - nobody except the two matching sessions can derive the key. The key might be proven to be indistinguishable from a random value.

Bonus: provides proofs where only either the KEM or the DH part is assumed secure.

2.5 full DH key exchanges

The protocol is given in Figure 5.

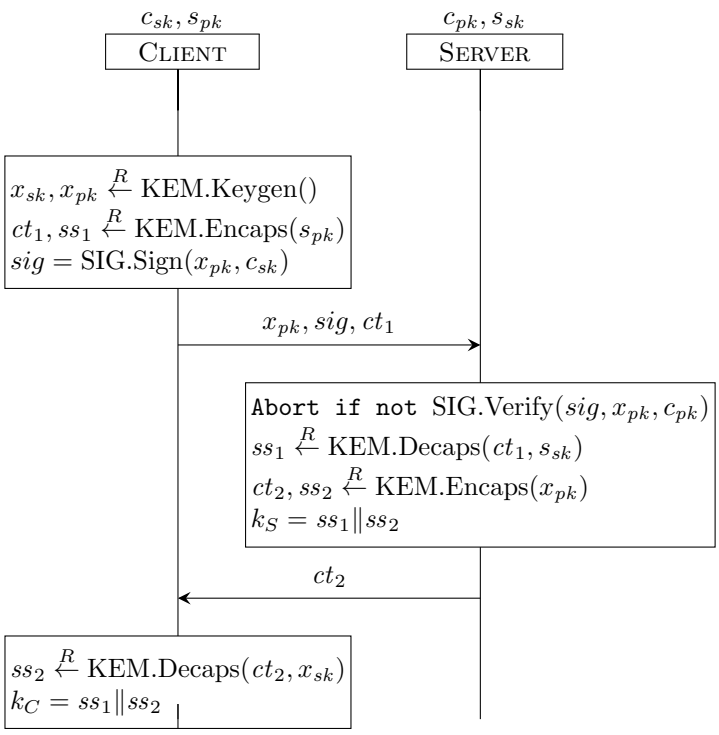

Figure 3: signed KEM key exchange

Description The server has a long term DH keypar s_{sk}, s_{pk}. This key is used to authenticate the server, while the client and the server exchange fresh DH values to derive a shared secret key.

Properties This protocol should provide two guarantees against an active machine-in-the-middle attacker:

- injective authentication - if a client derives a key with some parameters (the set of public keys), a corresponding server session derived the same key with the same parameters.
- secrecy - nobody except the two matching sessions can derive the key. The key might be proven to be indistinguishable from a random value.

A Attack traces

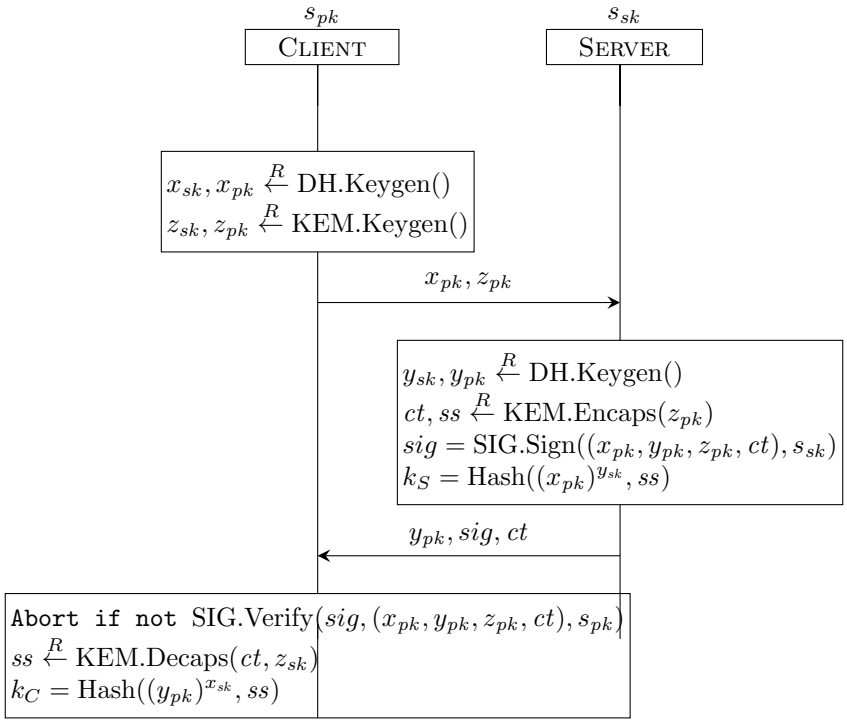

Figure 4: Hybridization of a signed Diffie-Hellman key exchange

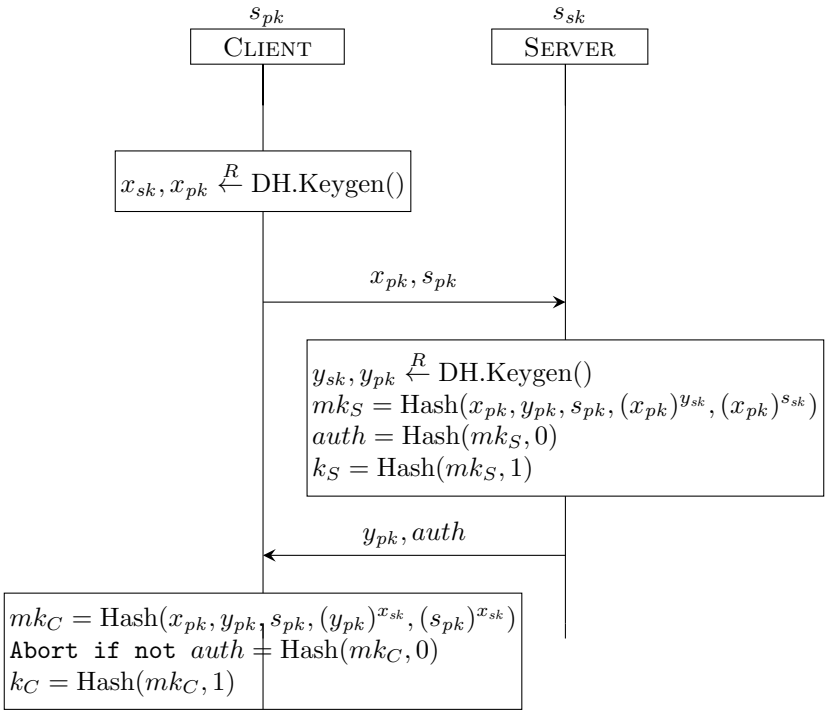

Figure 5: Simplified NTOR

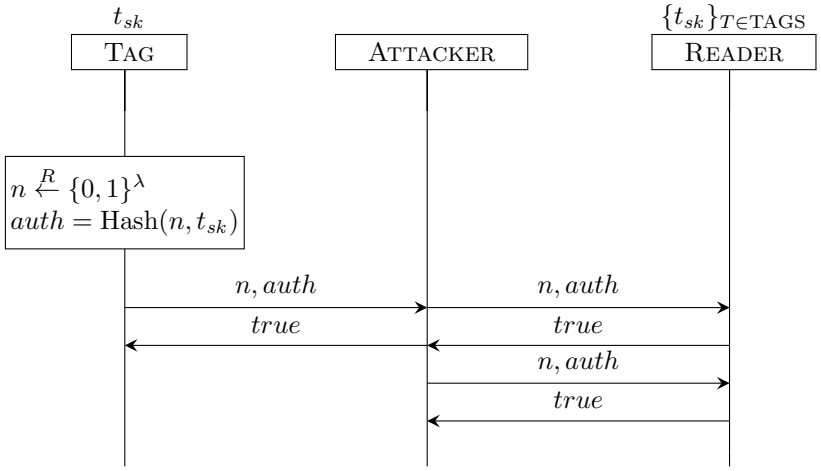

Figure 6: Basic Hash - basic replay attack breaking injective authentication

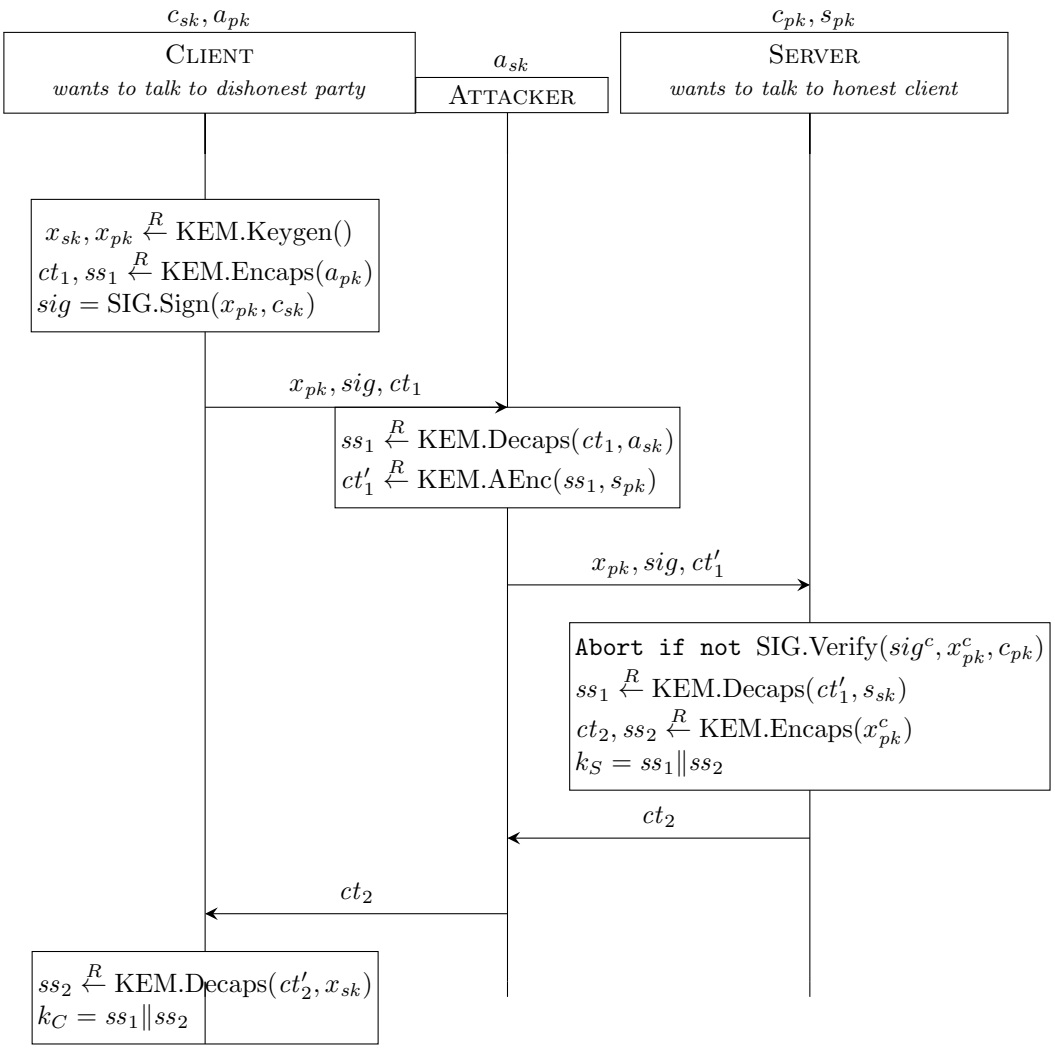

Figure 7: signed KEM key exchange - Unknown Key-share Attack through reencapsulation attack .
Both honest client and honest server derive the same key, but do not believe to be talking together.

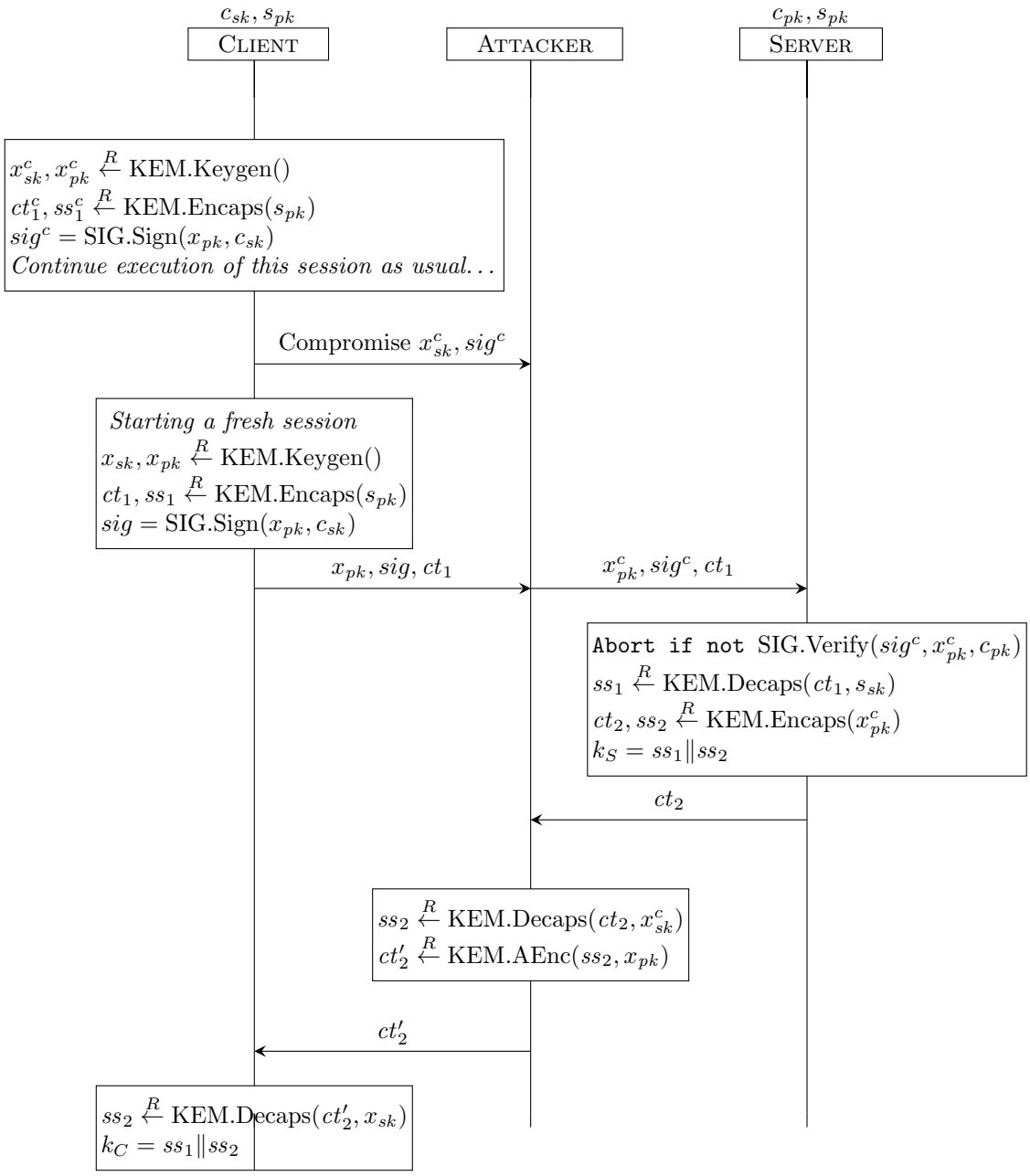

Figure 8: signed KEM key exchange - Reencapsulation attack against session independence.
The attacker learns half of the key of a fresh session by compromising material from an independent session.