Selfie: reflections on TLS 1.3 with PSK

Nir Drucker and Shay Gueron

University of Haifa, Israel,
and
Amazon, Seattle, USA

Abstract. TLS 1.3 allows two parties to establish a shared session key from an out-of-band agreed Pre Shared Key (PSK). The PSK is used to mutually authenticate the parties, under the assumption that it is not shared with others. This allows the parties to skip the certificate verification steps, saving bandwidth, communication rounds, and latency. We identify a security vulnerability in this TLS 1.3 path, by showing a new reflection attack that we call "Selfie". The Selfie attack breaks the mutual authentication. It leverages the fact that TLS does not mandate explicit authentication of the server and the client in every message.
The paper explains the root cause of this TLS 1.3 vulnerability, demonstrates the Selfie attack on the TLS implementation of OpenSSL and proposes appropriate mitigation.
The attack is surprising because it breaks some assumptions and uncovers an interesting gap in the existing TLS security proofs. We explain the gap in the model assumptions and subsequently in the security proofs. We also provide an enhanced Multi-Stage Key Exchange (MSKE) model that captures the additional required assumptions of TLS 1.3 in its current state. The resulting security claims in the case of external PSKs are accordingly different.

Keywords: TLS 1.3 · Selfie Attack · Reflection attack · Network security · Multi-Stage Key Exchange model

1 Introduction

TLS 1.3 (Hereafter TLS) [18] offers several options for secure key establishment. Some options mutually authenticate the communicating parties to each other, where a common handshake includes the verification of the server and the client through certificates. Another option uses a Pre Shared Key (PSK) that was agreed upon beforehand, either during a preceding handshake (i. e., a resumption PSK) or distributed out-of-band (i. e., *external* PSK). When a PSK is used, the underlying assumption is that a party receiving a message (that passes verification with the PSK) knows that the message was sent by a party that (also) owns that PSK. TLS allows to use this implicit authentication instead of certificate verification in order to save bandwidth and latency and also to support 0-RTT mode. In particular, this mode can be used for a network of communicating peers, where every node can act (in parallel) as server and as client (e. g., proxy servers and P2P communications).

We claim that this mode of TLS opens the door for a vulnerability: the sender of an authentic message could also be, under an attack scenario, the receiver itself. This situation lends itself to what is known as a reflection attack, which we call a “Selfie” attack. We describe this attack in the following sections.

Remark 1. When TLS 1.3 uses external PSKs it has no control on how these PSKs were generated and what entities have access to these keys. In this paper, we assume that PSKs are: a) generated as uniform random strings, and their lengths are appropriate for use in the cryptographic primitives (e.g., 256 bits); b) the distribution of the PSKs assigned a unique key for every pair of peers in the network (where every peer can act as both a client and a server). In theory, the network owner (”Dealer”) can choose to share PSKs differently (e. g., share across a group of peers). The implications of such behavior are discussed in Sections 7 and 9.

The paper is organized as follows. Section 2 describes our notation and illustrates one basic TLS 1.3 flow with PSKs. The Selfie attack is described in Section 3. Our demonstration is detailed in Section 4. Section 5 discusses other scenarios for the Selfie attack. In Section 6 we explain why the Selfie attack was not captured in previous security models. We describe several modes of distributing PSKs in Section 7. Some mitigation approaches are detailed in Section 8. “Group authentication” protocols are discussed in Section 9. We provide some details about our disclosure process in Section 10. Section 11 conclude this paper. Finally, our modified MSKE security model for TLS 1.3 is provided in Appendix A.

2 Preliminaries

2.1 Notation and Conventions

A string of bits of length l (bits) is denoted by $s[l-1:0]$. The length is denoted by $|s| = l$. Concatenation of the strings s_1 and s_2 is denoted by $s_1||s_2$. The notation $\perp$ is used hereafter for notating a protocol failure or an empty string. Uniform random sampling from a set W is denoted by $w \overset{\$}{\leftarrow} W$. Hexadecimal values are denoted with a `0x` prefix (e. g., `0x1F` is 31 decimal). A string of 128 bits is called a block. Let $H : \{0,1\}^* \to \{0,1\}^{\ell_H}$ be a hash function that is agreed during the handshake protocol (default to SHA256). The HMAC function is as defined in [11].

2.2 TLS 1.3

TLS (TLS 1.3) [18] supports various protocol flavors. For simplicity, we present here only the external PSK mode (i. e., where the PSK is not obtained from a previous handshake). We omit the asymmetric parts that are used for further authentication and the ticketing mechanism used for session resumption and use the terminology of [18]. The entities in TLS are clients and servers, where a client initiates the handshake and the server is the responder. Denote by

$$\mathsf{TH}(M_1,\ldots,M_n) = H(M_1||M_2||\ldots||M_n)$$

the transcript hash, where H is the negotiated hash function, and $M_1,\ldots,M_n$ are handshake messages.

The protocol. A client initiates the communication by sending a ClientHello (CH) message with: a) a list of supported ciphersuites (AEAD and HKDF); b) *supported groups* and signature algorithms (ignored here); c) a list of identities of the PSKs and a Key Exchange (KE) mode (either *psk_ke* or *psk_dhe_ke*). In addition, the client can send a `key_share` extension that includes a DH/ECDH ephemeral key. In *psk_dhe_ke* mode, the client must send this extension in order to achieve Forward Secrecy (FS). In *psk_ke* mode, the client can choose to send the `key_share` extension in order to allow the server to fallback into a normal (without PSKs) handshake. If the server accepts the KE mode, a specific PSK from the list, and a specific ciphersuite, it sends a ServerHello (SH) message with a `pre_shared_key` extension that identifies the selected PSK. If it selects to use the *psk_dhe_ke* mode, it must add a `key_share` extension with the server ephemeral key share. In addition, the server sends a Server Finished (SF) message $(\mathsf{HMAC}(\mathsf{finished\_key},\mathsf{TH}(\cdot))$.

If the server cannot complete the handshake with the requested parameters it chooses one of: a) abort; b) fall back to a full handshake; c) reply with a HelloRetryRequest (HRR) message. Cases (b) and (c) require asymmetric keys and are thus ignored here. The handshake ends after the client verifies the data received from the server. In this case, it may send a Client Finished (CF) message $(\mathsf{HMAC}(\mathsf{finished\_key},\mathsf{TH}(\cdot))$ back to R.

Remark 2. When the client sends the server a list of PSKs every PSK is associated with a binder. The binder is computed by $(\mathsf{HMAC}(\mathsf{binder\_key},\mathsf{TH}(\cdot)))$. In the **external** PSK mode the hash function is either predefined (together with the PSK) or defaults to SHA256.

3 The Selfie attack

Consider a network of communicating peers, where each node acts as a TLS server and as a TLS client. Assume that PSKs are pre-distributed and that the chosen TLS mode uses these PSKs without certificates.

Figure 1 illustrates the Selfie attack on such network, launched by an active eavesdropper (Eve) that traps the communication between two legitimate parties (Alice and Bob). The attack leverages two properties of TLS in this setting:

1. Alice can open parallel independent connections.
2. Alice as client does not explicitly check the identity of the server. She only verifies that the server is a legitimate owner of the relevant PSK. However, she cannot rule out the possibility that the parties that run the server and

the client are the same identities (i. e., that it receives an echo of her own messages).

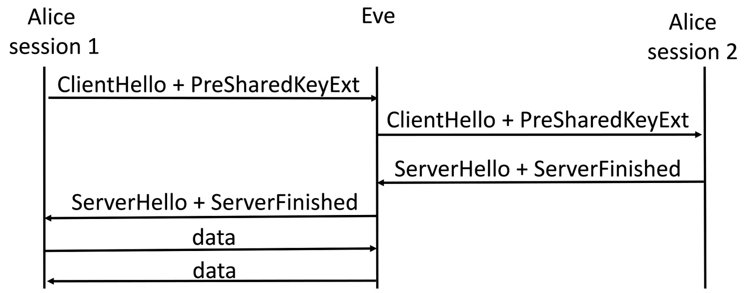
Alice
session 1
Eve
Alice
session 2
ClientHello + PreSharedKeyExt
ClientHello + PreSharedKeyExt
ServerHello + ServerFinished
ServerHello + ServerFinished
data
data

Fig. 1. The Selfie Attack. Eve tricks Alice to believe she is talking to Bob while she is actually talking with herself. See description in the text.

The attack

1. Alice sends Bob the CH message with a `pre_shared_key` extension.
2. Eve captures the message and echoes it back to Alice, pretending (implicitly) to be Bob.
3. Alice in-fact receives a Selfie image of what she had sent to Bob. After Alice authenticates this message, she is tricked to believe that it was sent from Bob (because "only Bob" has the PSK that allows him to send a correctly authenticated CH message).
4. Alice replies (to Bob) with SH+SF messages.
5. Eve captures these messages and echoes them back to Alice.
6. At this point Alice has opened a Selfie session with herself.
7. After the session established, Alice sends data (presumably to Bob) and receives the same data back (presumably coming from Bob).

This sequence constitutes a Man in the Middle (MITM) attack on Alice, that breaks the TLS claimed properties of [18][Appendix E]:

> "Peer authentication: The client's view of the peer identity should reflect the server's identity. If the client is authenticated, the server's view of the peer identity should match the client's identity."

Implications. To illustrate the threat that this Selfie attack poses, consider the following scenario.

Alice opened a session, presumably with Bob, but in reality with herself. Subsequently, Alice sends a message to (fake) Bob:

```
"If you have the file data.txt you can delete it. I hold a copy."
```

However, this message is echoed back to Alice (instead of reaching Bob) as if coming from Bob. At this point, Alice is led to believe that Bob sent her the message of the instructions

```
"If you have the file data.txt you can delete it. I hold a copy."
```

Alice checks that she has a copy of `data.txt` (which, of course, she has) and deletes the file. If there is no other copy of `data.txt`, the file is lost.

4 Demonstrating the Selfie attack

This section describes a demonstration of the Selfie attack in a way that it can be repeated by the reader. For completeness we also describe the system that we use for the experiments: a Linux (Ubuntu 16.04.3 LTS) OS running on a platform equipped with the latest 7^{th} Generation Intel® CoreY^{TM} processor (”Kaby Lake”) - Intel® Xeon® Platinum 8124M CPU at 3.00 GHz Core® $i5 - 750$.

The smallest network configuration for the Selfie attack requires at least one node that acts as a server and as a client (Alice) and a switch that acts as the Selfie mirror (Eve). Our experiment was executed on a single desktop machine as follows. We emulated a virtual network using Mininet [15]. To run its virtual machine image we used VirtualBox [17]. Inside the virtual machine we installed the latest version of OpenSSL [16] configured to enable TLS 1.3.

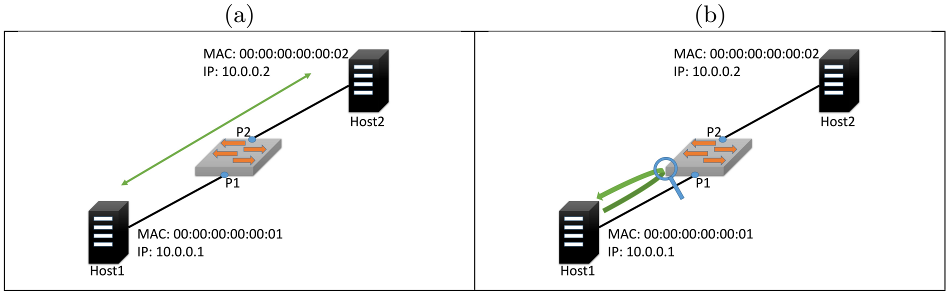

Fig. 2. Network configuration. (a) normal configuration; both hosts communicate with each other (as intended); (b) the Selfie attack configuration: all packets sent to port P1 are reflected back to the sender, where the MAC and IP addresses are swapped.

We started the virtual network inside the virtual machine by executing

```
sudo mn −−topo single ,2 −−mac −−switch ovsk
```

This generates a network with two nodes (Host 1 is Alice and Host 2 is Bob) and an *ovsk* switch (Eve) as illustrated in Figure 2. We used two configurations for the switch in order to simulate the normal intended operation (Figure 2 panel a) and the Selfie attack scenario (panel b). The associated command lines for the normal configuration, where packets from port 1 (P1) are forwarded to port 2 (P2) and vice versa are

```
sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

For the Selfie attack configuration. We use the commands

```
sudo ovs-ofctl add-flow s1 priority=100,in_port=1,
               actions=mod_dl_src:00:00:00:00:00:02,
               mod_dl_dst:00:00:00:00:00:01,
               mod_nw_src:10.0.0.2,
               mod_nw_dst:10.0.0.1,output:in_port
```

and

```
sudo ovs-ofctl add-flow s1
               priority=1000,dl_type=0x806,
               nw_proto=1,actions=flood
```

The first command reflects every packet that arrives to P1 back to its origin (which is Host 1). However, note that the source and destination (IP and MAC address) are flipped. The second command tells the switch how to handle Address Resolution Protocol (ARP) requests. It is important (for this experiment) to set the priority of the second command to be higher than the priority of the first command. This allows ARP replies (otherwise, the second host is unidentified and will not receive any ARP messages).

In both hosts we set the PSK to have the (arbitrary) value

```
export PSK=11111111111111112222222222222222
           33333333333333334444444444444444
```

Now we opened a TLS 1.3 server (with OpenSSL) on both hosts that are configured to listen to port 1443 as follows

```
apps/openssl s_server -accept 1443
                      -tls1_3 -4
                      -ciphersuites TLS_AES_128_GCM_SHA256
                      -psk $PSK
                      -nocert
                      -num_tickets 0
```

Subsequently, we opened a client on Host 1 with the command

```
apps/openssl s_client -connect 10.0.0.2:1443
                      -tls1_3 -4
                      -psk $PSK
                      -ciphersuites TLS_AES_128_GCM_SHA256
```

Remark 3. We comment about the specific TLS 1.3 implementation of OpenSSL. Here, the client always offers the *psk_dhe_ke* KE mode to the server. The server prefers the *psk_dhe_ke* mode over the *psk_ke* mode (because it provides FS). Therefore, our demonstration shows an attack on TLS 1.3 with external PSK in the *psk_dhe_ke* KE mode. Clearly, the Selfie attack is also possible in the *psk_ke* mode (see details in Section 5).

The results

In the normal mode, the operation was as intended: Host 1 is communicating with Host 2 and the TLS 1.3 with PSK session was established correctly. By contrast, under the Selfie attack, Host 1 ended up communicating with itself consuming exactly the same messages that it delivered. The implications were discussed above.

Remark 4. It is interesting to note that this experiment cannot be repeated with BoringSSL (and not OpenSSL) as the underlying cryptographic library. While BoringSSL enables TLS 1.3 by default, in the client and server, it does not support (implement) the option of using PSK without certificates.

5 Other selfish scenarios that are prone to the Selfie attack

The Selfie attack also applies to the following TLS cases:

- TLS 1.2. This can also be demonstrated using the above experiment. We believe that situations where TLS 1.2 is used with PSKs are not common. Therefore, our focus is on TLS 1.3.
- TLS 1.3 with PSKs in *psk_ke* mode (without FS). The Selfie attack applies also to the same flow without ephemeral keys. To demonstrate the Selfie attack, we first prepared a patched version of OpenSSL that disable the *psk_dhe_ke* mode for the TLS client. We rerun the demonstration with the patched OpenSSL while we added the `-allow_no_dhe_kex` flag to the client and the server commands.

Remark 5. We provide some details on the OpenSSL implementation of TLS. The TLS 1.3 server application (`s_server`) of OpenSSL provides two flags `-no_dhe` and `-allow_no_dhe_kex` with the following documented description "Disable ephemeral DH" and "In TLS v1.3 allow non-(ec)dhe based key exchange on resumption", respectively. Therefore, we expect that using these flags will cause the server to operate only in the *psk_ke* mode. However, this did not give the expected results because the client always offers the *psk_dhe_ke* mode. We could not find a (intuitive) way to run the client in *psk_ke* mode. Therefore, we patched the client code to disable the *psk_dhe_ke* mode. This allows us to demonstrate that the Selfie attack is valid also in the *psk_ke* mode and not only in the *psk_dhe_ke* mode as above. Note that the attack relies on a property of TLS and not on a specific implementation of OpenSSL.

Some possible scenarios where the external PSKs may be used, and the Selfie attack could be relevant are:

- Content Delivery Network (CDN) nodes that act as proxies are considered a target for the Selfie attack. Here, an attacker can reflect a request from a CDN node to itself, poison its caches with errors which can lead to Denial of Service (DoS) attack.
- P2P networks.
- WiFi networks that rely on one PSK for the entire network.
- Leader-election and consensus protocols.

6 Where did the security proofs miss?

The Selfie attack comes as a surprise, since TLS 1.3 is the first TLS version that was designed alongside with security proofs, e. g., the studies in [3, 5–9, 12–14]. This is even more surprising since the Selfie attack is actually a manifestation of a reflection attack that is already known in other contexts. Here, we try to understand how this slipped through.

Consider the papers [8,9] that focus on `draft-ietf-tls-tls13-14` (hereafter, `draft-14`). They show the security of 0-RTT and PSK flows under the MSKE model [7], augmented for the replayable 0-RTT keys case. This model follows the game-based paradigm [2], and assumes that the adversary controls the network and communications. However, the proof provided in [8,9] considers the use of PSKs only for session resumptions. The gap is that the proof ignores the case of external PSKs, which is also a valid TLS scenario. Note that external PSKs were introduced at least as early as `draft-7` ("TLS provides a pre-shared key (PSK) mode ... e.g., a key established out of band"). We can now follow the proof itself. To prove the Match game, property number 4 on TLS (i. e., "Sessions are partnered with the intended (authenticated) participant and (for mutual authentication) share the same key index.") [8, 9] claim

> "As honest sessions only used their own pre-shared secret identifier *psk_id*, this value included (via ClientHello) in all session identifiers ensures agreement of both the intended partner and key index."

This argument fails exactly in the case of external PSKs. The reason is that a PSK belongs to at most two parties, but the protocol cannot tell which one of the two parties is sending a message. This leads to the Selfie attack.

The analysis in [4, 14] uses an automatic tool to prove the security of TLS (`draft-10` and `draft-21`) namely, Tamarin [1]. It considers the option of using "out-of-band" PSKs, but leaves some options for future work [19] as follows:

> "In future work, it would be interesting to write precise authentication properties to understand the nature of implicit authentication for out-of-band PSK authentication"

The gap in the model that was introduced to Tamarin is the assumption that a PSK cannot be shared by more than two parties (one client and one server). Clearly, this is not the case in our attack scenario, where we assume that a PSK is unique per host (node) that can run a client and a server (that share the PSK). Consequently, the PSK is shared among more than two (sub)-entities (e. g., two servers and a client as in the demonstration in Section 4).

Apparently, theoretical proofs and automatic-tools' proofs did not account for this (somehow hidden, but still allowable) case of TLS 1.3. Fortunately, the mitigation is simple, and we also see that some cryptographic libraries as BoringSSL did not even implemented this TLS option.

7 Dealing with the Dealer

The common (informal) definition of TLS with PSKs is "TLS is a KE protocol between two parties that hold a PSK **pss**". However, this is not the case when external PSKs are used. Therefore, a better (informal) definition is: "TLS is a KE protocol between two parties that hold a PSK **pss** that is shared by a set $G_{\mathbf{pss}}$ of parties, with **at least** two members." Hereafter, every **pss** is associated with some group $G_{\mathbf{pss}}$.

The PSKs can be negotiated by the parties or be distributed by some trusted authority called a Dealer. The Dealer's policy for distributing the keys among the parties is beyond the scope of the TLS protocol. We assume that the Dealer is trusted and that it keeps the PSKs secret, according to the following policies.

1. The Dealer distributes a unique key for every session between a client-server pair. An example for this scenario is the resumption mode (with tickets).
2. The Dealer distributes the keys in a way that for every client-server pair it hands a unique key (that can be used in multiple sessions). In this case, replay attacks may be possible.
3. The Dealer divides the network entities (clients and servers in all nodes) into groups (not necessarily disjoint) and distributes the same PSK to all members of the same group. Different groups are given different keys. This policy

reduces to case 1 (or case 2) if all the groups have exactly two members (the groups are not necessarily disjoint). Therefore, hereafter we discuss case 3 when at least one group contains three members. An example for this scenario is the current TLS 1.3 definition. We label it as a *group_authentication* protocol.

In our attack scenario the Dealer distributes the same PSK to two nodes, where each node is a client and a server. Overall, four entities (two clients and two servers) end up with the same PSK.

Case 1 is captured in previous security models and is defined as a *mutual _authenticated* protocol. Case 2 is not captured in the MSKE model [8]. But we believe that TLS with this type of a Dealer is secure (perhaps without replay protection). Case 3 can lead to the Selfie attack and other problems.

8 Mitigating the Selfie attack

One mitigation is to modify the TLS protocol [18] to include the following restriction:

External PSKs MUST be used together with server certificates.

This prevents the Selfie attack. However, the problem is that PSKs are introduced to the protocol in order to avoid certificates[1]. Therefore a better option, is to limit the use of external PSKs in TLS by adding the following statement to its definition.

A PSK MUST NOT be shared between more than one client and one server.

This prevents the Selfie attack, without the need for certificates. We recommend this solution.

9 Group authenticated protocol

This section discusses the third policy of the Dealer (*group_authentication* protocols). This policy is the current state of TLS 1.3 and we recommend to remove

[1] We point out that server certificates may be spared here: even adding a client certificate would prevent the Selfie attack (note that a recent IETF draft [10] further suggests to include also the client certificate in the handshake but with a different motivation).

it (see above). To show why, we study the security properties and security models that apply to this case, even if the Selfie attack is prevented.

A *group_authentication* protocol, has a new set of security properties, that require a different assumption on the trustworthiness of the entities (i. e., the adversary capabilities): one must accept the fact that in every group with three (or more) members, every member can impersonate the other members in the group (and of course, decrypt and modify messages if the session keys are derived without ephemeral DH keys).

Another property (similar to *mutual_authentication* protocols) is that there is no protection against an adversary that controls a node (this is captured in the Corrupt query of the MSKE model as explained in Appendix A). This implies that the protocol is secure only if all the nodes in the network are secure (and trusted), and the adversary can only control the network.

However, unlike *mutual_authentication* protocols, a network adversary $\mathcal{A}$ against *group_authentication* protocols has additional attack vectors for gathering confidential information leaked during the protocol execution. For example, $\mathcal{A}$ can intercept a CH message sent from a node n_1 to a node n_2 together with some early (0-RTT) data. It can then forward this traffic to both n_2 and n_3 (both in the same group of n_1) and measure their response time. If the execution of the protocol (or the processing of the early data) is not done in a side channel protected way, $\mathcal{A}$ may learn something about the state (caches and memory) of n_2 and n_3. In addition, if the early data is not idempotent it may be executed twice (on different nodes). To prevent this attack the CH message must include an authenticated n_2 identity that will prevent $\mathcal{A}$ from forwarding the message to n_3.

Preventing the Selfie attack. Preventing the Selfie attack in *group_authentication* protocols can be done in at least three ways, but we only consider the third option to be practical.

1. Preventing parallel sessions (on the same node).
2. Every party (client or server) caches all the nonces that it generated, and rejects a connection from other parties if they use a cached nonce.
3. Every participating party gets (during the setup of the network) a unique identity that is known to all the other parties in the network. Then, the identities of the sender and the intended receiver are included in every handshake message and also authenticated. The client and the server must verify the validity of the claimed identities.

Mitigating the Selfie attack by using option one is obviously undesired. Option 2 involves the overhead of maintaining a cache and the overhead of sharing the cache between several processes on the same node. In addition, both options do not solve the leaking information attack described above. Therefore, we recommend the third option, adding identities, which in TLS can be done by enforcing the use of the Server Name Identification (SNI) extension.

10 Responsible disclosure

Our demonstrated attack was performed on the most updated versions of the cryptographic libraries evaluated, as published at the time of discovery.

We are not aware of any usage of PSK's in a way that opens the door to a vulnerability, but out of an abundance of caution, we followed the practice of responsible disclosure. We disclosed our findings in March 2019 to Akamai, Apple, Citrix, CloudFlare, Google, Microsoft, OpenSSL, Oracle, and also to Eric Rescorla of Mozilla (handling TLS 1.3 definitions).

11 Discussion

It is interesting to recall that TLS 1.3 is the first TLS version that was developed alongside with security proofs, both manual proofs and also proofs that leverage automatic tools. The protocol was therefore considered (under the appropriate assumptions) to be provably secure. Nevertheless, we see here that there is a gap in the proofs, because one (allowable) usage option in TLS 1.3 is susceptible to the Selfie attack. Although the attack scenario does not fall in the main intended usage of TLS, it is not an inconceivable scenario. Indeed, in a network of nodes that send and receive messages (i. e., can play the role of a server and a client) under the same PSK (e. g., CDN proxies, P2P networks, and WiFi communications), there is no reason a priori to use different keys for sending and for receiving packets.

In general, a security protocol should capture, warn, or at least mention all the assumptions it relies on, in order to protect implementers from misunderstandings or mistaken usages. For example, we note the TLS 1.3 specification explicitly uses different keywords (e. g., MUST, MAY, SHOULD) to specify requirements and restrictions.

While some constraints such as "keys MUST be kept confidential" are considered trivial and thus omitted, it is important that other assumptions are made explicit. Specifically, the Selfie attack is the result of an undocumented assumption in TLS 1.3 (as explained above).

The security of a cryptographic protocol can be proved under different models (e. g., MSKE) that may make different assumptions and conclude different properties. Of course, all of the protocol assumptions must be stated and also be included in the model if one wishes to obtain correct results (see Section 6). Proofs that are produced by automatic tools may be incorrect or incomplete if the underlying model does not capture all the assumptions or if its details are not fed correctly. In our case, we identify two independent problems: a) TLS 1.3 does not explicitly forbid the *group_authentication* mode, which allows the Selfie attack; b) the models that were used in the proofs did not account for this scenario, and therefore mistakenly concluded that the protocol is secure in all of its variants. To illustrate the necessary modifications to the MSKE model that cover the relevant assumptions, we outline the enhanced model in Appendix A. We point out that the proof that TLS is secure under this model and the properties of a *group_authentication* protocol, is similar to the proof in [8]. Of course, the resulting security statement is different.

The Selfie attack is possible simply because TLS 1.3 allows the *group_authentication* mode. However, we note that a problem still remains in practice *even if* the protocol is changed to forbid this use case. The reason is that users of cryptographic libraries such as OpenSSL typically believe/assume that all of the options supported by the library are secure. On the other hand, the library itself cannot enforce this restriction and prevent it from being used. One way to avoid this problem is to not support the use of external PSKs in TLS. Note that BoringSSL does exactly that. For libraries that want to support all of the TLS options (e. g., OpenSSL), this approach may not be acceptable. We therefore propose to disable the external PSK mode by default, and enable it only through a compilation flag, with an explicit warning and documentation.

Acknowledgments

We thank Matt Campagna, Adam Langley, Colm MacCarthaigh, Kenny Paterson, and Eric Rescorla, for useful discussions and suggestions. We thank Gilad Ram for recommending Mininet for the demonstration.

This research was supported by: The Israel Science Foundation (grant No. 1018/ 16); The BIU Center for Research in Applied Cryptography and Cyber Security, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office; the Center for Cyber Law & Policy at the University of Haifa, in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office.

References

1. Tamarin prover. https://tamarin-prover.github.io/#
2. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) Advances in Cryptology — CRYPTO' 93. pp. 232–249. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
3. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 483–502. IEEE (2017). https://doi.org/10.1109/SP.2017.26
4. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A Comprehensive Symbolic Analysis of TLS 1.3. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1773–1788. CCS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134063
5. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol. Tech. rep. (2017)
6. Fischlin, M., Günther, F., Schmidt, B., Warinschi, B., Gunther, F., Schmidt, B., Warinschi, B.: Key Confirmation in Key Exchange: A Formal Treatment and Implications for TLS 1.3. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 452–469 (2016). https://doi.org/10.1109/SP.2016.34

7. Fischlin, M., Günther, F.: Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1193–1204. CCS '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2660267.2660308
8. Fischlin, M., Günther, F.: Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates. Tech. rep. (2017)
9. Fischlin, M., Günther, F.: Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 60–75 (2017). https://doi.org/10.1109/EuroSP.2017.18
10. Housley, R.: TLS 1.3 Extension for Certificate-based Authentication with an External Pre-Shared Key. Internet-Draft draft-ietf-tls-tls13-cert-with-extern-psk-00, Internet Engineering Task Force (Feb 2019), https://datatracker.ietf.org/doc/html/draft-ietf-tls-tls13-cert-with-extern-psk-00, work in Progress
11. Hugo, K., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. https://tools.ietf.org/html/rfc2104 (February 1997)
12. Krawczyk, H., Wee, H.: The OPTLS Protocol and TLS 1.3. pp. 81–96. IEEE (2016). https://doi.org/10.1109/EuroSP.2016.18
13. Li, X., Xu, J., Zhang, Z., Feng, D., Hu, H.: Multiple Handshakes Security of TLS 1.3 Candidates. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 486–505 (2016). https://doi.org/10.1109/SP.2016.36
14. van der Merwe, T.: An Analysis of the Transport Layer Security Protocol Thyla van der Merwe. Ph.D. thesis, Royal Holloway, University of London (2018), http://www.isg.rhul.ac.uk/{~}kp/theses/TvdMthesis.pdf
15. Mininet: Mininet - An Instant Virtual Network on your Laptop (or other PC) version mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64.zip. http://mininet.org/ (2019)
16. OpenSSL: OpenSSL commit 38023b87f037f4b832c236dfce2a76272be08763 (February 2019), https://github.com/openssl/openssl/commit/38023b87f037f4b832c236dfce2a76272be08763
17. Oracle: VirtualBox 5.1 (2018), https://www.virtualbox.org/
18. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (aug 2018). https://doi.org/10.17487/RFC8446, https://rfc-editor.org/rfc/rfc8446.txt
19. Scott, S.: TLS 1.3 modelled in Tamarin. https://samscott89.github.io/TLS13_Tamarin/ (2018)

A A modified Multi-Stage Key Exchange (MSKE) model

Our modification of the MSKE model [7] augmented for the replayable 0-RTT keys case [8,9] in order to support *group_authentication* protocols (third policy in Section 7) is described below. This model follows the game-based paradigm [2], and assumes that the adversary controls the network and communications (and thus has the power to read and modify the protocol messages).

Remark 6. As mentioned above in a *group_authentication* protocol, with a PSK $\mathbf{pss}$ and entities $e_1, e_2, e_3 \in G_{\mathbf{pss}}$, e_3 can impersonate e_2 to e_1 or it can read-/write/modify all the traffic transferred between e_1 and e_2. In case ephemeral keys are supported then e_3 can still impersonate e_2 to e_1 but it cannot read-/write/modify the traffic after the ephemeral keys were exchanged.

Remark 7. Our recommendation is to forbid the group authentication mode from the TLS protocol. Nevertheless, for completeness we outline the modified model below. The proof that TLS is secure under this model is almost identical to the proof in [8] and thus omitted.

The modification we propose addresses the current TLS state where group authentication is allowed. The main changes in our modified Multi-Stage Key Exchange (MSKE) model is that we add the *group_authentication* authentication property and the $\mathsf{DupSecret}$ and $\mathsf{SpreadSecret}$ queries. The $\mathsf{DupSecret}$ query allows to use a PSK in more than one session. The $\mathsf{SpreadSecret}$ query allow to share a PSK **pss** with a new party. This query can be called only if all the sessions that use **pss** are *group_authentication* sessions.

A.1 The model

The modified MSKE model is described below, where for brevity, only the parts that are relevant to Section 2.2 are mentioned (see [8] for the full model[2]). The model [9] uses 4 parameters M, AUTH, USE, and REPLAY to describe a protocol.

- $\mathsf{M} \in \mathbb{N}$: the number of stages (equals the number of keys). Hereafter, for every vector X, its per-stage value is denoted by X_i, $i \in \{1, \ldots, \mathsf{M}\}$.
- $\mathsf{AUTH} \in \{unauth, unilateral, mutual, \textbf{group\_authenticated}\}^{\mathsf{M}}$: a set of supported authentication properties per-stage. The values indicate that a) I and R are unauthenticated; b) only R is authenticated (to I); c) I and R are mutually authenticated (to each other); d) I and R are mutually authenticated to be in the same group G (where $I, R \in G$).
- $\mathsf{USE} \in \{internal, external\}^{\mathsf{M}}$: USE_i indicates whether the i-th stage key is used internally (during the handshake) or not. External keys cannot be used internally, but internal keys can be used externally.
- $\mathsf{REPLAY} \in \{replayable, nonreplayable\}^{\mathsf{M}}$: $REPLAY_i$ indicates whether the i-th stage is replayable so a party can easily force identical communication and thus identical session identifiers and keys.

Let Chal be the game challenger and $\mathcal{U}$ the set of participants. Uniquely identify every session in the game with a label $(U, V, k) \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$ that indicates the k-th local session of U (the session owner) and V. Every session is associated with a (not necessarily unique) PSK **pss** associated with a unique PSK identifier **psid**. Chal maintains the vectors $\overrightarrow{\mathbf{pss}}_{U,V}$ and $\overrightarrow{\mathbf{psid}}_{U,V}$ of PSKs generated by $\mathcal{A}$, where the k-th entry refers to the k-th **pss** and the corresponding **psid**, respectively, shared by U and V. The main difference from the model of [8] is that a pair (**pss**, **psid**) may be repeated for different labels (U, V, k). To this end, Chal maintains two new maps $\overrightarrow{\mathbf{psid}}_G$ and $\overrightarrow{\mathbf{psid}_{\mathbf{pss}}}$, where $\overrightarrow{\mathbf{psid}}_G[\mathbf{psid}] = G_{\mathbf{pss}}$ and $\overrightarrow{\mathbf{psid}_{\mathbf{pss}}}[\mathbf{psid}] = \mathbf{pss}$. Note that for protocols with PSKs, $\overrightarrow{\mathbf{pss}}_{U,V} = \overrightarrow{\mathbf{pss}}_{V,U}$. However, this is not the case when a protocol involves

[2] Modifying the model when ephemeral keys are involved is done in the same way and therefore omitted.

asymmetric keys. At the beginning of the game, Chal chooses $b_{test} \xleftarrow{\$} \{0,1\}$. In addition, a list List holds per-session tuples with the following information (for brevity, default values are written in **boldface**).

- **label** $\in \mathsf{LABELS}$: a unique label.
- $U \in \mathcal{U}$: the session owner.
- $V \in \big(\mathcal{U} \cup \{*\}\big)$: U's partner. The protocol may use the "unknown identity" symbol "*" to indicate that the partner identity will be set (only once) in a later stage of the protocol.
- **role** $\in \{\mathit{initiator}, \mathit{responder}\}$: the role of the session owner.
- **auth** $\in \mathsf{AUTH}$: the (per stage) intended authentication type.
- $\mathbf{st_{exec}} \in (\mathbf{RUNNING} \cup \mathit{ACCEPTED} \cup \mathit{REJECTED})$: the state of execution, where $\mathit{RUNNING} = \{\mathit{running}_i \mid i \in \mathbb{N}_0\}$, $\mathit{ACCEPTED} = \{\mathit{accepted}_i \mid i \in \mathbb{N}\}$, $\mathit{REJECTED} = \{\mathit{rejected}_i \mid i \in \mathbb{N}\}$.
- **stage** $\in \{\mathbf{0}, \ldots, \mathsf{M}\}$: the current protocol stage, it is incremented to i when $\mathbf{st_{exec}}$ reaches $\mathit{accepted}_i$ or $\mathit{rejected}_i$.
- **sid** $\in \big(\{0,1\} \cup \{\bot\}\big)^{\mathsf{M}}$: a vector of session identifiers.
- **cid** $\in \big(\{0,1\} \cup \{\bot\}\big)^{\mathsf{M}}$: a vector of contributive identifiers.
- $K \in \big(\{0,1\} \cup \{\bot\}\big)^{\mathsf{M}}$: a vector of session keys.
- $\mathbf{st_{key}} \in \{\mathbf{fresh}, \mathit{revealed}\}^{\mathsf{M}}$: the session keys' state.
- **tested** $\in \{\mathit{true}, \mathbf{false}\}^{\mathsf{M}}$: indicates whether K_i has been tested or not.
- $k \in \mathbb{N}$: the PSK index, for U and V.
- **pss** $\in \big(\{0,1\}^* \cup \{\bot\}\big)$: the PSK.
- **psid** $\in \big(\{0,1\}^* \cup \{\bot\}\big)$: the PSK identifier.

key dependence. The term key dependence means that if K_i, $i < \mathsf{M}$ is disclosed before the key K_{i+1} was generated and K_{i+1} depends on K_i, then K_{i+1} is compromised. This property is captured in the Reveal query. Note that the model assumes that K_{i+1} is not trivially dependent on K_i (e. g., $K_{i+1} = H(K_i)$). Rather, it is still indistinguishable from random given the revealed K_i.

Let $\mathcal{A}$ be a Probabilistic Polynomial-Time (PPT) adversary that controls the network. It can intercept, inject, and drop messages. A protocol provides stage-k FS if keys from the k-th stage on are forward secrets. The model includes a *lost* flag (initialized to false) indicating that $\mathcal{A}$ trivially lost (e. g., if it corrupts a tested instance). $\mathcal{A}$ interacts with the protocol by using the following queries:

$\mathsf{NewSecret}(U,\ V,\ k,\ \mathbf{psid})$**:** Generate a fresh **pss** with identifier **psid**. If **psid** has been already registered to another **pss** or that $\overrightarrow{\mathbf{pss}}_{U,V}[k] \neq \bot$, return $\bot$. Otherwise, set $\overrightarrow{\mathbf{pss}}_{U,V}[k] = \overrightarrow{\mathbf{pss}}_{V,U}[k] = \mathbf{pss}$ and $\overrightarrow{\mathbf{psid}}_{U,V}[k] = \overrightarrow{\mathbf{psid}}_{V,U}[k] = \mathbf{psid}$. These values form the k-th secret between parties U and V. Set $\overrightarrow{\mathbf{psid}}_G[\mathbf{psid}] = \{U, V\}$ and $\overrightarrow{\mathbf{psid}_{\mathbf{pss}}}[\mathbf{psid}] = \mathbf{pss}$

$\mathsf{SpreadSecret}(U,\ \mathbf{psid},\ \{(V, k_V)\}_{V \in \overrightarrow{\mathbf{psid}}_G[\mathbf{psid}]})$**:** If $\overrightarrow{\mathbf{psid}}_G[\mathbf{psid}] = \bot$ or if $U \in \overrightarrow{\mathbf{psid}}_G[\mathbf{psid}]$, return $\bot$. If there exists $1 \leq i \leq \mathsf{M}$ and a session s with $s.\mathbf{psid} = \mathbf{psid}$ and $s.\mathbf{auth}_i \neq \mathit{group\_authentication}$ return $\bot$. For every V if $\overrightarrow{\mathbf{pss}}_{U,V}[k_V] \neq \bot$, return $\perp$. Otherwise set $\overrightarrow{\mathbf{pss}}_{U,V}[k_V] = \overrightarrow{\mathbf{pss}}_{V,U}[k_V] = \overrightarrow{\mathbf{psid}_{\mathbf{pss}}}[\mathbf{psid}]$ and $\overrightarrow{\mathbf{psid}}_{U,V}[k_V] = \overrightarrow{\mathbf{psid}}_{V,U}[k_V] = \mathbf{psid}$. Finally, add U to $\overrightarrow{\mathbf{psid}}_{G}[\mathbf{psid}]$.

DupSecret(U, V, k, $\bar{k}$): If $\overrightarrow{\mathbf{pss}}_{U,V}[k] =\perp$ or $\overrightarrow{\mathbf{pss}}_{U,V}[\bar{k}] \neq\perp$, return $\perp$. Otherwise, set $\overrightarrow{\mathbf{pss}}_{U,V}[\bar{k}] = \overrightarrow{\mathbf{pss}}_{V,U}[\bar{k}] = \overrightarrow{\mathbf{pss}}_{U,V}[k]$ and $\overrightarrow{\mathbf{psid}}_{U,V}[\bar{k}] = \overrightarrow{\mathbf{psid}}_{V,U}[\bar{k}] = \overrightarrow{\mathbf{psid}}_{U,V}[k]$.

NewSession(U, V, **role**, **auth**, k): If $\overrightarrow{\mathbf{pss}}_{U,V}[k] =\perp$ or $(|\overrightarrow{\mathbf{psid}}_{G}[\mathbf{psid}]| \geq 2$ and $\mathbf{auth} \neq$ *group_authentication*) return $\perp$. Otherwise, create a new session with a (unique) new label $\mathbf{label} = (U, V, k)$, U with role **role**, V, *auth*, $\mathbf{pss} = \overrightarrow{\mathbf{pss}}_{U,V}[k]$, $\mathbf{psid} = \overrightarrow{\mathbf{psid}}_{U,V}[k]$, and **auth**. Add (**label**, U, V, **role**, **auth**, k, **pss**, **psid**) to List and return **label**.

Send(**label**,m): Find a session s that is identified by **label**. If it does not exist return $\perp$. Otherwise, run the protocol on behalf of U or V on the message m. Return the response and the updated execution state $s.\mathbf{st}_{\mathbf{exec}}$. Specifically, when $s.\mathbf{role} =$ *initiator* use $m =$ "*init*" to initiate the protocol. If during the protocol execution, $s.\mathbf{st}_{\mathbf{exec}}$ is changed to $accepted_i$ for some i, then

- immediately suspend the execution and return $accepted_i$ to $\mathcal{A}$.
- if there exits a partnered session $s' \in$ List ($s.\mathbf{sid}_i = s'.\mathbf{sid}_i$) with $s'.\mathbf{st}_{\mathbf{key}_i} = revealed$, for key-independent protocols set $s.\mathbf{st}_{\mathbf{key}_i} = revealed$. For key-dependent protocols set $s.\mathbf{st}_{\mathbf{key}_{i'}} = revealed$, $i' \geq i$.
- if there exits a partnered session $s' \in$ List with $s'.\mathbf{tested}_i = true$, set $s.\mathbf{tested}_i = true$ and (only if $\mathsf{USE}_i = internal$) set $s.K_i = s'.K_i$.
- if the intended communication partner $V \neq *$ is corrupted, set $s.\mathbf{st}_{\mathbf{key}_i} = revealed$.

Subsequently, $\mathcal{A}$ can call a Send(**label**, "continue") query to ask Chal to resume executing the protocol. The returned answer is the expected answer from the protocol execution.

Reveal(**label**, i): for the i-th stage of session s, identified by **label**, reveal $s.K_i$. If no such s exists or $s.\mathbf{stage} < i$, or $s.\mathbf{tested}_i = true$, return $\perp$. Otherwise, set $s.\mathbf{st}_{\mathbf{key}_i} = revealed$ and return $s.K_i$. For every partnered session $s' \in$ List with $s'.\mathbf{stage} \geq i$, set $s'.\mathbf{st}_{\mathbf{key}_i} = revealed$. In the case of keys-dependent protocols, future keys might be depend on the revealed key, if $s.\mathbf{stage} = i$, set $s.\mathbf{st}_{\mathbf{key}_{i'}} = s'.\mathbf{st}_{\mathbf{key}_{i'}} = revealed$ for all $i' > i$ and s' such that $s'.\mathbf{stage} = i$. Note that if $s'.\mathbf{stage} > i$ then the keys $s'.K_{i'}$, derived in the partnered session, are not considered to be revealed by this query.

Corrupt(**psid**): Provide $\mathcal{A}$ with $\overrightarrow{\mathbf{psid}_{\mathbf{pss}}}[\mathbf{psid}]$. No further queries are allowed to any session s with $s.\mathbf{psid} = \mathbf{psid}$. For consistency, for every session s with $s.\mathbf{psid} = \mathbf{psid}$ and $i \in \{1, \ldots, \mathsf{M}\}$ (in the non-FS case) or $i < j$ or $i > s.\mathbf{stage}$ (in the case of stage-j FS), call Reveal($s.\mathbf{label}$, i). The latter refers to session keys before enabling FS or that have not yet been established. The calls to the Reveal queries are internal. Their responses and their fact that they return $\perp$ in case of a call with $i > s.\mathbf{stage}$ are ignored.

$\mathsf{Test}(\mathbf{label},\ i)$**:** for the i-th stage of the session s identified with **label**, perform a test on $s.\mathbf{st}_{\mathbf{key}_i}$.

- If no such s exists or $s.\mathbf{st_{exec}} \neq accepted_i$ or $s.\mathbf{tested}_i = true$, return $\perp$.
- If there is a partnered session $s' \in \mathsf{List}$ with $s'.\mathbf{st_{exec}} \neq accepted_i$, set $lost = true$.
- If $s.\mathbf{auth}_i = unauth$ or if $s.\mathbf{auth}_i = unilateral$ and $s.\mathbf{role} = responder$, but there is no session $s' \neq s \in \mathsf{List}$ with $s.\mathbf{cid}_i = s'.\mathbf{cid}_i$, then set $lost = true$.

In any other case, set $s.\mathbf{tested}_i = true$. Set $K_1 = s.K_i$ and $K_0 \overset{\$}{\leftarrow} \mathcal{D}$, where $\mathcal{D}$ is the session key distribution. If $\mathsf{USE}_i = internal$, set $s.K_i = K_{b_{test}}$ (because it is also used for consistent future deployments within the key exchange protocol). For every partnered session $s' \in \mathsf{List}$, $s.\mathbf{st_{exec}} = s'.\mathbf{st_{exec}} = accepted_i$) set $s'.\mathbf{tested}_i$ to true and only if $USE_i = internal$ set $s'.K_i = s.K_i$ for consistency. Return $K_{b_{test}}$ to $\mathcal{A}$.

A.2 General security properties of the MSKE model

The MSKE model captures the leak of PSKs or the leak of session keys in the Corrupt or the Reveal queries, respectively. However, it does not model the leak of internal values or session's state. This is because the adversary is only a network adversary and not a node adversary.

The MSKE model includes two security games: a) the Match security game - ensuring that both parties in a session have the same session identifiers (**sid**); b) the Multi-Stage security game that ensures key secrecy (as defined by Bellare-Rogaway). In our enhanced MSKE mdoel, the Match security game $G^{Match}_{KE,\mathcal{A}}$ is defined as follows.

Definition 1 (Match security). *Let KE be a MSKE protocol with properties (*M*,* AUTH*,* USE*,* REPLAY*). Let $\mathcal{A}$ be a PPT adversary interacting with KE via the above queries in the following $G^{Match}_{KE,\mathcal{A}}$ game:*

$\mathcal{A}$ accesses the NewSecret*,* DupSecret*,* SpreadSecret*,* NewSession*,* Send*,* Reveal*, and* Corrupt *oracles and at some point stops without any output. $\mathcal{A}$ wins the game ($G^{Match}_{KE,\mathcal{A}} = 1$) if for two partnered sessions s, s', where $s.\mathbf{sid}_i = s'.\mathbf{sid}_i \neq \perp$ and $i \leq \mathsf{M}$ one of the following ocurres*

1. *$s.\mathbf{stage}_i \neq rejected$ but $s.K_i \neq s'.K_i$.*
2. *$s.\mathbf{auth}_i \neq s'.\mathbf{auth}_i$.*
3. *$s.\mathbf{cid}_i \neq s'.\mathbf{cid}_i$ or $s.\mathbf{cid}_i = s'.\mathbf{cid}_i = \perp$.*
4. *$s.\mathbf{role} = initiator$, $s'.\mathbf{role} = responder$, and*
   - *$s.\mathbf{auth}_i = s'.\mathbf{auth}_i = unilateral$, but $s.V \neq s'.U$*
   - *$s.\mathbf{auth}_i = s'.\mathbf{auth}_i = mutual\_authentication$, but $s.V \neq s'.U$ or $s.U \neq s.V$*
   - *$s.\mathbf{auth}_i = s'.\mathbf{auth}_i = group\_authentication$, but one of $s.U, s.V, s'.U, s'.V \notin \overrightarrow{\mathbf{psid}}_G[s.\mathbf{psid}]$.*

   or $s.\mathbf{psid} \neq s'.\mathbf{psid}$.

In addition, $\mathcal{A}$ wins the game if 5) there exist two sessions s, s' and $i \neq j$ such that $s.\mathbf{sid}_i = s'.\mathbf{sid}_j \neq \perp$; 6) there exist sessions $s_1 \neq s_2 \neq s_3$, $i \leq \mathsf{M}$, $\mathsf{REPLAY}_i = nonreplayable$ such that $s_1.\mathbf{sid}_i = s_2.\mathbf{sid}_i = s_3.\mathbf{sid}_i \neq \perp$.

A KE is Match-secure (with a security parameter λ) if for all PPT adversaries $\mathcal{A}$:

$$Adv_{KE,\mathcal{A}}^{Match} := Pr\left[G_{KE,\mathcal{A}}^{Match} = 1\right] \leq negl(\lambda)$$

Note that the difference between our enhanced Match definition (that supports *group_authentication* protocols) and the original MSKE Match definition is in item 4.

The Multi-Stage security game $G_{KE,\mathcal{A}}^{Multi-Stage}$ is defined as follows and is the same in the enhanced and the original MSKE model.

Definition 2 (Multi-Stage security). *Let KE be a MSKE protocol with properties (*M, AUTH, USE, REPLAY*) and a PPT adversary $\mathcal{A}$ interacting with KE via the above queries in the following $G_{KE,\mathcal{A}}^{Multi-Stage}$ game:*

Chal *chooses $b_{test} \overset{\$}{\leftarrow} \{0,1\}$ and sets lost $= false$. Subsequently, $\mathcal{A}$ accesses the* NewSecret, DupSecret, SpreadSecret, NewSession, Send, Reveal, Test, *and* Corrupt *oracles and at some point stops and outputs a guess b.* Chal *sets the lost $= true$ if there exist two sessions s, s' and $i \leq \mathsf{M}$ such that $s.\mathbf{sid}_i = s'.\mathbf{sid}_i, s.\mathbf{st}_{\mathbf{key}_i} = revealed$, and $s'.\mathbf{tested}_i = true$. ($\mathcal{A}$ has tested and revealed the key in a single session or in two partnered sessions.) $\mathcal{A}$ wins the game ($G_{KE,\mathcal{A}}^{Multi-Stage,\mathcal{D}} = 1$), if $b = b_{test}$ and lost $= false$.*

KE is Multi-Stage-secure in a key-dependent resp. key-independent and non-FS resp. stage-j FS manner with concurrent authentication types AUTH, *key usage* USE, *and replayability property* REPLAY *if KE is Match-secure and for every PPT adversary $\mathcal{A}$:*

$$Adv_{KE,\mathcal{A}}^{Multi-Stage,D} := Pr\left[G_{KE,\mathcal{A}}^{Multi-Stage,D} = 1\right] - \frac{1}{2} < negl(\lambda)$$