$\mathcal{S}$ between U and V. Then W participates in two additional sessions $\mathcal{S}_1$ and $\mathcal{S}_2$ with V and U, respectively. In these two sessions, W transmits values s_U and s_V that are copied from $\mathcal{S}$. (Of course, W does not know the exponents r_U and r_V corresponding to s_U and s_V, respectively.) Then, after the sessions $\mathcal{S}_1$ and $\mathcal{S}_2$ have concluded, W requests the keys for these two sessions, which is permitted in a known session key attack.

The session keys K, K_1, and K_2 for the sessions $\mathcal{S}$, $\mathcal{S}_1$, and $\mathcal{S}_2$ (respectively) are as follows:

$$\begin{aligned} K &= \alpha^{r_U a_V + r_V a_U} \\ K_1 &= \alpha^{r_U a_V + r'_V a_W} \\ K_2 &= \alpha^{r'_U a_W + r_V a_U}. \end{aligned}$$

Given K_1 and K_2, W is able to compute K as follows:

$$K = \frac{K_1 K_2}{(s'_V s'_U)^{a_W}}.$$

Therefore this is a successful known session key attack.

The triangle attack can also be defeated through the use of a key derivation function, as described above. It is conjectured that this modified version of *MTI/A0* is secure against known session key attacks.

12.5 Deniable Key Agreement Schemes

The concept of ***deniability*** provides an interesting counterpoint to the idea of non-repudiation, which is a central requirement of signature schemes. Recall that non-repudiation (in the context of a signature scheme) means that someone who has signed a message cannot later plausibly deny having done so. This is useful in any context where a signature should be considered to be a binding commitment, such as signing a contract.

On the other hand, there are situations where Alice and Bob might wish to engage in a private conversation, but neither of them desires that any third party should be able to prove that they had a particular conversation, even if the keys used to encrypt that conversation are leaked at some later time. In other words, the conversation is secure, but it affords plausible deniability to the participants in the event of a future key compromise.

Informally, a key agreement scheme is said to be ***deniable*** if this property is achieved when the resulting session keys are used to encrypt a conversation between the parties executing the key agreement protocol. To be more precise, we consider the following scenario.

1. An adversary gains access to the private keys belonging to V (this adversary could be V himself, or some third party.

Protocol 12.4: X3DH KAS

The public domain parameters consist of a group $(G, \cdot)$, an element $\alpha \in G$ having order n, and a key derivation function denoted by **KDF**.
Each user T has a long-term private key a_T, where $0 \leq a_T \leq n-1$, and a corresponding long-term public key

$$b_T = \alpha^{a_T}.$$

The value b_T is included in T's certificate and is signed by the *TA*.

1. U chooses r_U at random, $0 \leq r_U \leq n-1$, and computes

   $$s_U = \alpha^{r_U}.$$

   Then U sends $\mathbf{Cert}(U)$ and s_U to V.

2. V chooses r_V at random, $0 \leq r_V \leq n-1$, and computes

   $$s_V = \alpha^{r_V}.$$

   Then V sends $\mathbf{Cert}(V)$ and s_V to U.
   Finally, V computes the session key

   $$K = \mathbf{KDF}({s_U}^{r_V} \| {s_U}^{a_V} \| {b_U}^{r_V}),$$

   where he obtains the value b_U from $\mathbf{Cert}(U)$.

3. U computes the session key

   $$K = \mathbf{KDF}({s_V}^{r_U} \| {b_V}^{r_U} \| {s_V}^{a_U}),$$

   where she obtains the value b_V from $\mathbf{Cert}(V)$.

At the end of the session, U and V have both computed the same session key

$$K = \mathbf{KDF}(\alpha^{r_U r_V} \| \alpha^{r_U a_V} \| \alpha^{r_V a_U}).$$

2. The adversary produces a purported transcript of a session of a key agreement scheme involving U and V.

3. The goal is to determine if the transcript constitutes evidence that the session between U and V actually took place. If so, this would implicate U.

For some key agreement schemes, it turns out that it is possible to simulate (or forge) a transcript that is identical to a real transcript. For such a scheme, there is

no way to determine if the session in question actually occurred. So the adversary is thus unable to implicate U and such a scheme would therefore be deniable.

To illustrate this concept of deniability, it is useful to contrast the basic Diffie-Hellman (Protocol 12.1), which is deniable, with *STS* (Protocol 12.2), which is not deniable.

First we look at basic *Diffie-Hellman*. Suppose for the purpose of discussion that U and V use Protocol 12.1 to derive a session key. Then, at some later time, V wishes to implicate U. V can store and reveal all the information he sent or received from U, along with his own private keys. In terms of the key agreement protocol, this information (i.e., the transcript) would consist of the following information:

- V's private key, a_V, and
- the public keys $b_U = \alpha^{a_U}$ and $b_V = \alpha^{a_V}$.

From this transcript, the key $K = \alpha^{a_U a_V}$ can be computed using the formula $K = {b_U}^{a_V}$. However, there is no convincing evidence that it was U who shared this key with V, because V could have simply created the public key b_U himself. The entire transcript could be forged by the adversary, and so we would say that basic *Diffie-Hellman* is deniable.

On the other hand, *STS* is not deniable. The reason for this is that both U and V sign the public keys they exchange during the protocol. Now the transcript would consist of

- V's private key, a_V,
- the public keys $b_U = \alpha^{a_U}$ and $b_V = \alpha^{a_V}$, and
- the signatures $\mathbf{sig}_U(ID(V) \parallel b_U \parallel b_V)$ and $\mathbf{sig}_V(ID(U) \parallel b_V \parallel b_U)$.

This transcript provides convincing evidence that the associated key $K = \alpha^{a_U a_V} = {b_U}^{a_V} = {b_V}^{a_U}$ was created in a session involving U and V. This because the public keys b_U and b_V, along with $ID(V)$, were signed by U. Therefore, V can implicate U, and U cannot plausibly deny that she took part in the given session of the key agreement protocol with V.

Thus, if deniability is a desired property of the key agreement scheme, then *STS* does not provide a satisfactory solution. On the other hand, basic *Diffie-Hellman*, while deniable, is susceptible to intruder-in-the-middle attacks. So the interesting question is how to design deniable key agreement schemes that are secure against intruder-in-the-middle attacks. We now present a recent method, known as *X3DH*, which is incorporated into the *Signal* messaging protocol.[3] *X3DH* is quite similar to *MTI* in some respects, but it uses three Diffie-Hellman keys instead of two. The *X3DH* key agreement scheme is presented as Protocol 12.4.

[3] *Signal* is a messaging protocol that has achieved widespread use since its development by Open Whisper Systems in 2013, notably in applications such as *WhatsApp*.

We do not discuss the security properties of *X3DH* in detail. However, we mention that *X3DH* is deniable and it provides perfect forward secrecy (see the Exercises). We also observe that a basic intruder-in-the middle attack does not succeed because the adversary cannot modify the long-term public keys without detection (since they are retrieved from a certificate). The adversary can modify the public keys s_U and s_V, changing them to s'_U and s'_V, respectively. However, in this situation, he would not be able to compute the resulting modified keys defined by the protocol. U would compute the key

$$\mathbf{KDF}(\alpha^{r_U r'_V} \parallel \alpha^{r_U a_V} \parallel \alpha^{r'_V a_U}).$$

However, the adversary cannot compute this key because he does not know the value of $\alpha^{r_U a_V}$. V would compute the key

$$\mathbf{KDF}(\alpha^{r'_U r_V} \parallel \alpha^{r'_U a_V} \parallel \alpha^{r_V a_U}).$$

The adversary does not know the value of $\alpha^{r_V a_U}$, so he cannot compute this key either.

12.6 Key Updating

Key updating schemes provide methods of updating keys on a regular basis. Ideally, the compromise of a key should not affect the security of previously-used keys (this is the "perfect forward secrecy" property, as defined in Section 11.1), nor should it allow the adversary to determine keys that are established in the future. One obvious way to approach this problem would be to execute a *Diffie-Hellman* KAS every time a message is sent. Each key is used only once and then deleted, and "new" keys have no dependence on old keys. Of course, *Diffie-Hellman* requires "expensive" operations such as exponentiations in finite fields, so we might seek less costly alternatives.

We already described the *Logical Key Hierarchy*, which is a type of key updating for dynamic networks, in Section 11.4. In Section 11.4, the reason for updating keys was to allow users to join or leave the network without impacting the security of the other network users. On the other hand, in this section, we have a pair of users who wish to communicate over a long period of time, and they wish to update their keys periodically.

We will now describe in simplified form some of the key updating techniques that are used in the *Signal* protocol. One of the goals of *Signal* is to provide ***end-to-end encryption***, which ensures that only the communicating parties can decrypt the encrypted communications. No third party, including the service provider, should have the technological capability to decrypt messages.

One design element incorporated into *Signal* is sometimes termed a ***Diffie-Hellman ratchet***. This manages to reduce the amount of key computation (as compared to setting up a new Diffie-Hellman key for every message sent) by about

25%. The idea is as follows: Every time U sends a message to V (or vice versa), the sender chooses new public and private keys (e.g., a_U and $b_U = \alpha^{a_U}$ in the case of user U) and sends the new public key along with a message that is encrypted under the old Diffie-Hellman key. The next Diffie-Hellman key to be used by the recipient is computed from the new public key along with the recipient's old private key. See Protocol 12.5 for the details of this protocol.

In practice, an authenticated version of *Diffie-Hellman* might be preferred. We are just describing the updating (i.e., ratcheting) process using basic *Diffie-Hellman* for simplicity.

If U and V used a *Diffie-Hellman KAS* to compute a new key every time a message is sent, they would each have to perform two exponentiations per message sent. Here, each party performs three exponentiations to compute two successive keys (after the initial key K_{00} is computed using two exponentiations by both parties). This is how the 25% speedup is achieved.

Observe that an adversary who manages to access a user's private key will only be able to use it to compute two successive keys.

The second major type of key updating or key ratcheting that is incorporated into *Signal* makes use of a key derivation function denoted by **KDF**. The function **KDF** has two inputs and two outputs. The two inputs are

1. a constant value C, and
2. a ***KDF key***, say K_i,

and the two outputs are

1. a "new" KDF key, say K_{i+1}, and
2. an ***output key***, denoted by OK_{i+1}.

We denote this process by the notation

$$\mathbf{KDF}(C, K_i) = (K_{i+1}, OK_{i+1}).$$

KDF is used to iteratively construct a ***KDF chain***. This requires an initial KDF key K_0. Then a sequence of output keys is produced as follows:

$$\begin{aligned}
\mathbf{KDF}(C, K_0) &= (K_1, OK_1) \\
\mathbf{KDF}(C, K_1) &= (K_2, OK_2) \\
\mathbf{KDF}(C, K_2) &= (K_3, OK_3),
\end{aligned}$$

etc. The output keys $OK_1, OK_2, \ldots$ are used to encrypt and decrypt messages.

A KDF chain is faster than a public key ratchet because it is based on a fast hash function. However, the security properties are weaker. An adversary who compromises a KDF key K_i (and who knows the value of the constant C) can compute all subsequent output keys, beginning with OK_{i+1}. (However, assuming that the function **KDF** is one-way, the adversary cannot compute any previous output

Protocol 12.5: DIFFIE-HELLMAN RATCHET

The public domain parameters consist of a group $(G, \cdot)$ and an element $\alpha \in G$ having order n.

1. U chooses a private key a_0 and computes a corresponding public key α^{a_0}. She sends α^{a_0} to V.

2. V chooses a private key b_0 and computes a corresponding public key α^{b_0}. He also computes the Diffie-Hellman key

$$K_{00} = (\alpha^{a_0})^{b_0}$$

and he sends α^{b_0} to U along with a message encrypted with K_{00}, say $y_{00} = e_{K_{00}}(x_{00})$.

3. U receives α^{b_0} and y_{00}. She computes the Diffie-Hellman key

$$K_{00} = (\alpha^{b_0})^{a_0}$$

and then she uses K_{00} to decrypt y_{00}. U then chooses a new private key a_1 and she computes the corresponding public key α^{a_1}. She sends α^{a_1} to V. Finally, U computes the Diffie-Hellman key

$$K_{10} = (\alpha^{b_0})^{a_1}$$

and she uses K_{10} to encrypt a message $y_{10} = e_{K_{10}}(x_{10})$. The value y_{10} is sent to V.

4. V receives α^{a_1} and y_{10}. He computes the Diffie-Hellman key

$$K_{10} = (\alpha^{a_1})^{b_0}$$

and then he uses K_{10} to decrypt y_{10}. V then chooses a new private key b_1 and computes the corresponding public key α^{b_1}. He sends α^{b_1} to U. Finally, V computes the Diffie-Hellman key

$$K_{11} = (\alpha^{a_1})^{b_1}$$

and he uses K_{11} to encrypt a message $y_{11} = e_{K_{11}}(x_{11})$. The value y_{11} is sent to U.

5. The preceding two steps are repeated as often as desired.

keys.) So a particular KDF chain should not be used for an extended period of time.

The *Signal* protocol uses both public-key ratchets and KDF chains. The combination of these two techniques is called the *double ratchet*. We do not go into the details. However, roughly speaking, the keys created in the public-key ratchet are not used to encrypt messages. They are instead used to initiate KDF chains. At any point in time, there are two active KDF chains that are maintained by U and V. U has a ***sending chain*** whose output keys are used to encrypt messages that U sends to V, and a ***receiving chain*** whose output keys are used to decrypt messages that U receives from V. V also has a sending chain and a receiving chain. The sending chain for V is identical to the receiving chain for U and the receiving chain for V is identical to the sending chain for U. Whenever the public-key ratcheting scheme is applied, it is used to derive two new initial KDF keys, one for each of these two KDF chains.

12.7 Conference Key Agreement Schemes

A ***conference key agreement scheme*** (or, *CKAS*) is a key agreement scheme in which a subset of two or more users in a network can construct a common secret key (i.e., a group key). In this section, we discuss (without proof) two conference key agreement schemes. The first CKAS we present was described in 1994 by Burmester and Desmedt. We also present the 1996 CKAS due to Steiner, Tsudik, and Waidner.

Both of these schemes are modifications of the *Diffie-Hellman KAS* in which m users, say $U_0, \ldots, U_{m-1}$, compute a common secret key. The schemes are set in a subgroup of a finite group in which the **Decision Diffie-Hellman** problem is intractable.

The *Burmester-Desmedt CKAS* is presented as Protocol 12.6. It is not hard to verify that all the participants in a session of this CKAS will compute the same key, Z, provided that the participants behave correctly and there is no active adversary who changes any of the transmitted messages. Suppose we define

$$Y_i = b_i^{a_{i+1}} = \alpha^{a_i a_{i+1}}$$

for all i (where all subscripts are to be reduced modulo m). Then

$$X_i = \left(\frac{b_{i+1}}{b_{i-1}}\right)^{a_i} = \left(\frac{\alpha^{a_{i+1}}}{\alpha^{a_{i-1}}}\right)^{a_i} = \frac{\alpha^{a_{i+1}a_i}}{\alpha^{a_{i-1}a_i}} = \frac{Y_i}{Y_{i-1}}$$

for all i. Then the following equations confirm that the key computation works