Lecture Notes for Advanced Web Security 2017

Part 4 — Secure Messaging

Martin Hell

1 Off-The-Record Messaging

Speaking off-the-record can have different meaning, but is often used in a situation where someone talks to a journalist and does not want the information to be either attributed to the source or that it should not be published at all. Looking at communication, if no security primitives are used it is not possible to protect it from eavesdroppers and there is no possibility to have message authentication and integrity, i.e., proof who sent the message and that it has not been modified on the channel. On the other hand, if cryptographic primitives are used it is much more difficult to later deny having said something.

Off-the-record messaging, first published in [3] and later improved in [2], provides a protocol to achieve the goals of such an informal information exchange between two parties having a face-to-face conversation in a private room, but in a setting where Alice and Bob communicates over the Internet using instant messaging. Alice will be certain that no one else can hear the conversation and Bob will later have no proof of what Alice has said. More specifically, the protocol aims to meet the following goals [1].

- Encryption. Messages are encrypted to prevent an eavesdropper from reading them.
- Authentication. Alice is certain that messages from Bob actually comes from Bob, and vice versa.
- Deniability. *After* the conversation, Alice can deny having sent the messages, i.e., there is no proof that Alice's messages were composed by her. The same holds for Bob's messages.
- Perfect forward secrecy. If your long-term secret is compromised, previous conversations are still secure.

Encryption and perfect forward secrecy are very straight-forward to obtain. Also, authentication and deniability are, by themselves, easy to obtain, but the combination of the two can seem a bit counter-intuitive. If you authenticate messages using e.g., a digital signature, then you can not later deny having sent them. This is a feature of digital signature as they provide non-repudiation.

The OTR protocol does not introduce much new technology, but it uses a few interesting techniques in order to fulfill some particular requirements. The description provided in this section will slightly differ from the actual protocol presented in [2, 3]. Implementation specific modifications are e.g., left out in order to instead highlight the main ideas and features. The discussion of the protocol will serve as a motivation for looking at cryptographic problems such as the socialist millionaires problem and malleability.

1.1 Authentication and Key Agreement

The first part of the protocol is the authentication and key agreement phase. In this phase, Alice and Bob are negotiating keys to use for encryption and message authentication. They also authenticate each other so that they are certain that no one else has access to the negotiated keys. In the first version of the OTR protocol, this was done using signed Diffie-Hellman messages. Alice and Bob each had a pair of long-term asymmetric keys. We denote Alice public key PK_A and her private key SK_A, and similar PK_B and SK_B for Bob. The key agreement messages are then given by

$$\begin{array}{lcll} A \to B & : & (g^{x_1})_{SK_A}, & PK_A \\ B \to A & : & (g^{y_1})_{SK_B}, & PK_B \end{array}$$

After verifying the signatures, Alice can compute the shared secret as $(g^{y_1})^{x_1}$ and Bob can compute the secret as $(g^{x_1})^{y_1}$. The key used in further communication is then computed as the hash of the shared value, $H(g^{x_1 y_1})$.

While this certainly works, and is very similar to how it is done in the Ephemeral Diffie-Hellman key exchange in TLS, it poses some unwanted restrictions in a messaging protocol intended for common use. The peer authentication in TLS require trusted public keys and a PKI, something that is more or less feasible to maintain and be part of for web services. However, this is much less practical for end users. Even in TLS, the end users are typically not authenticated as this is often left to the application layer after the authentication and key agreement phase of TLS. The problem was also evident in the SET protocol. In order to make this practical, Alice and Bob must exchange fingerprints of their public key and verify this fingerprint in order to authenticate the key exchange messages. Without the digital signatures, Eve than mount a Man-in-the-Middle (MitM) attack on the protocol and agree on one key with Alice, and one with Bob. She could then add, delete, or modify messages as she acts as a MitM.

In newer versions of the OTR protocol, the verification of the public key fingerprint uses the fact that Alice and Bob, since they know each other, likely already have some low entropy shared secrets. It is likely that Alice can ask Bob a question that only Bob can answer, e.g., "which movie did we watch last night", or "where did we find that dead parrot". A secure exchange using such low entropy secrets can be achieved using a solution to the *Socialist Millionaire Problem*.

1.2 Socialist Millionaire Problem

Consider two millionaires who wish to determine who is richest, but they do not want to reveal any information to each other about their wealth. This problem is called the *Millionaires Problem* and was introduced and solved in [7]. A variant of this problem is the Socialist Millionaires Problem [5], in which they wish to determine if their wealth is equal, but not disclosing any information about their wealth. Obviously, if it is equal, both will know the wealth of the other, but if it is not equal no one has any information about the wealth of the other.

In the protocol, Alice and Bob both have a secret, x and y respectively. The goal is to determine whether $x = y$. A very simple protocol for this would be to exchange the values $H(x)$ and $H(y)$. If $x = y$, then clearly $H(x) = H(y)$, but if $x \neq y$ then Alice would still not know Bob's value and vice versa. This, however, leaks information that can be used to brute force the other party's value. If Alice knows $H(y)$ she can just test inputs until she finds y' such that $H(y) = H(y')$ and she will know with overwhelming probability that $y = y'$. With low entropy values for x and y, this simple protocol will not solve the problem. Thus, a different solution is needed.

The protocol described here, which is also used in the implementation of the OTR protocol, is the one first given in [4]. It is based on the difficulty of the Diffie-Hellman problem, which in turn is related to, but not equivalent to, the discrete log problem. The Diffie-Hellman problem states that given g, p, g^x and g^y it is infeasible to find g^{xy}.

In the protocol, Alice and Bob agree on three different generators. They use a well known generator g_1 in order to agree on two new generators g_2 and g_3. This is agreed on by using a normal Diffie-Hellman exchange. Thus, we have

$$g_2 = g_1^{a_2 b_2} \bmod p, \qquad g_3 = g_1^{a_3 b_3} \bmod p$$

In the following, all operations will be mod p. Alice now picks a random value a and computes

$$(P_a, Q_a) = (g_3^a, g_1^a g_2^x)$$

Bob chooses his own value b and similarly computes

$$(P_b, Q_b) = (g_3^b, g_1^b g_2^y)$$

Now, all values P_a, Q_a, P_b and Q_b are exchanged. Using these values, Alice can use her a_3 to compute R_a and Bob can use his b_3 to compute R_b as follows

$$R_a = (Q_a Q_b^{-1})^{a_3}, \qquad R_b = (Q_a Q_b^{-1})^{b_3},$$

These two values, R_a and R_b, are exchanged and both can now compute

$$R_{ab} = R_a^{b_3} = R_b^{a_3}$$

Now they know that

$$\begin{aligned} R_{ab} &= \left(Q_a Q_b^{-1}\right)^{a_3 b_3} \\ &= \left(g_1^{(a-b)} g_2^{(x-y)}\right)^{a_3 b_3} \\ &= g_3^{a-b} g_2^{(x-y)a_3 b_3} \\ &= \left(P_a P_b^{-1}\right)\left(g_2^{a_3 b_3}\right)^{(x-y)} \end{aligned}$$

Thus, the final step is to check whether $R_{ab} = P_a P_b^{-1}$. If $x = y$ the equality will hold, but if $x \neq y$ it will not hold. It is also not possible for Alice or Bob to brute force the value $x - y$ in the exponent since $g_2^{a_3 b_3}$ is an unknown value. Each choice of $x - y$ will have a possible choice for $g_2^{a_3 b_3}$ such that the equation

$$R_{ab} \cdot \left(P_b P_a^{-1}\right) = \left(g_2^{a_3 b_3}\right)^{(x-y)}$$

will hold. For a complete proof of the security of this solution to the socialist millionaire problem, refer to [4]

1.3 Socialist Millionaire Applied to OTR

The group used in OTR is the Diffie-Hellman group 5, which is defined in RFC 3526 [6]. The prime p is 1536 bits and the generator g_1 as used in Section 1.2 is 2. In order for Alice and Bob to verify that they have the correct public key fingerprint of the other, the finger print is concatenated with the low entropy secret and the negotiated secret Diffie-Hellman value. This is hashed in order to produce their values x and y used in the socialist millionaire protocol,

$$x = y = H(PK_A \parallel PK_B \parallel g^{x_1 y_1} \parallel \textit{"shared secret"})$$

It is clear from Section 1.2 that an eavesdropper can not brute force the shared secret. Morevoer, for Eve to successfully act as a MitM she will have to guess the secret in the first attempt. Otherwise, if Eve fails, Alice and Bob will see that the protocol has failed. It can be noted that x and y are not computed exactly like this in the implementation of the protocol, but it still gives the idea.

1.4 Encryption and Authentication of Messages

In itself, Diffie-Hellman key exchange can be said to provide perfect forward secrecy since there is no data exchanged that is protected by public keys. This can be compared to a key exchange algorithm where an RSA public key is used to transport a secret from Alice to Bob. If Alice's private key is compromised, an attacker who has recorded previous key exchanges would also be able to decrypt the keys. On the other hand, for Diffie-Hellman, if one private exponent used in the key exchange is compromised, the resulting shared secret would also be

subject to a similar attack. To protect against this, OTR includes a new Diffie-Hellman key exchange message for each message that is sent. Thus, the forward secrecy is not only true for sessions, but for basically every single message. Messages are then composed as follows:

$$\begin{array}{lcll} \vdots & & & \\ A \to B & : & g^{x_i}, & E(M_j, k_{i-1,i-1}) \\ B \to A & : & g^{y_i}, & E(M_{j+1}, k_{i,i-1}) \\ A \to B & : & g^{x_{i+1}}, & E(M_{j+2}, k_{i,i}) \\ B \to A & : & g^{y_{i+1}}, & E(M_{j+3}, k_{i+1,i}) \end{array}$$

The keys used for encryption are given by a hash of the negotiated secret, $k_{ij} = H(g^{x_i y_i})$. Since digital signatures provide non-repudiation, a MAC is instead used to authenticate messages. A MAC uses symmetric keys so any MAC protected message sent by Alice to Bob can be verified by Bob, but he can not convince anyone else that it was created by Alice. It could just as well have been created by Bob. On the other hand, it can only have been created by Alice or Bob, not e.g., Charlie. The OTR protocol takes this one step further to allow anyone to create valid encrypted and authenticated messages. For the encryption part, a stream cipher is used. Recall that for a (binary additive) stream cipher, the ciphertext is computed as $c_i = m_i \oplus k_i$, where c_i, m_i and k_i are ciphertext bits, plaintext bits and keystream bits respectively. This encryption has an important property in that it is possible to make changes to the ciphertext, such that the two corresponding plaintexts are related in some known way. Encryption schemes with this property are called *malleable*. A stream cipher by itself is always malleable since flipping a ciphertext bit will correspond to flipping a plaintext bit since

$$c_i \oplus 1 = m_i \oplus k_i \oplus 1 = m_i \oplus 1 \oplus k_i.$$

The particular stream cipher used in OTR is AES in counter mode. Note that AES is a block cipher, but counter mode transforms it into a stream cipher by encrypting a counter and XORing the output with the plaintext. So, anyone that knows or can guess parts of the plaintext can also make any modification to this part of the message in order to create any message that they like. This of course assumes that they can authenticate the messages using the MAC key. To achieve this, OTR simply publishes the MAC keys as soon as they have been used. For each message a new MAC key is used and when a key has been used and the message has been verified by Bob, the key is simply added to the next message sent. Now, anyone would be able to create a message and authenticate it using the key. Thus, immediately after the MAC key has been used it is made practically worthless by publishing it.

References

[1] Off-the-Record Messaging. Available at: *https://otr.cypherpunks.ca/index.php*.

[2] Chris Alexander and Ian Goldberg. Improved user authentication in off-the-record messaging. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 41–47, New York, NY, USA, 2007. ACM.

[3] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84. ACM, 2004.

[4] Fabrice Boudot, Berry Schoenmakers, and Jacques Traor. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111:2001, 2001.

[5] Markus Jakobsson and Moti Yung. *Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers*, pages 186–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[6] T. Kivinen and M. Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526 (Proposed Standard), May 2003.

[7] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.