

# Perl Implementation of OTR

Deian Stefan

May 4, 2008

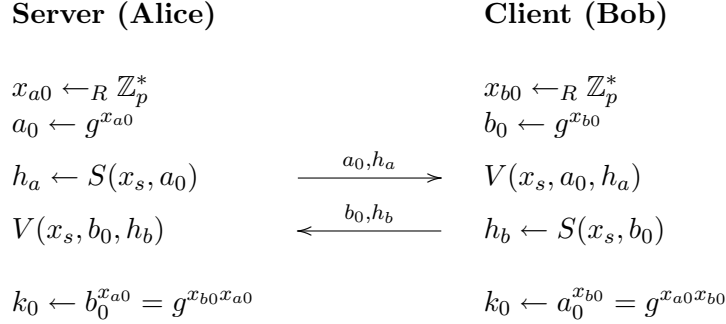
## Abstract

In this short paper the Off-the-Record (OTR) communication protocol is discussed. The communication protocol allows for parties to share an encrypted channel with *perfect forward secrecy* and *repudiability* – providing confidentiality between the communicating parties and in this implementation: plausible deniability to third parties. Methods for authentication, ciphertext integrity and malleable encryption are also discussed. The implementation allows the A slightly modified version of the protocol was implemented for a simple chat program in Perl using sockets. The code can be easily modified to use the protocol for other applications such as data transfer and multiple clients chat.

## 1 Initial Key Exchange and Authentication

The key exchange is based on the Diffie Hellman (DH) Algorithm, which is secure under the assumption of the Discrete Logarithm Problem (DLP). In the following,  $g, p$  are very large primes (on the order of 100 digits) shared between the two parties (Alice and Bob). Because it is easy to carry out a man in the middle attack on the DH protocol (see Appendix I), it is important for the parties to authenticate each other – validate that the received public keys ( $a_0$  for Bob and  $b_0$  for Alice) are from the party they trust. Although the OTR specifications specify the use of a digital signature scheme to sign the initial keys, these protocols do not allow for the parties to deny that they were a part of the conversation, so in this implementation the initial authentication requires the parties to share a common *secret* key  $x_s$  in order to sign the DH public keys – a Message Authentication Code (MAC) is used to sign ( $S$ ) and verify ( $V$ ) the keys. Even if the initial key exchange is eavesdropped, the third party (Eve) cannot prove that Alice sent Bob a message (or vice versa) since there

is no public-key aspect in the exchange.



Because user *passwords* are *not* random and therefore not secure it is important to use a one-way function ( $H_s$ , which is SHA-256 (Secure Hash Algorithm) in this implementation) to *hash* the shared  $x_s$  and use that as the key for the signing and verification algorithms (Algorithm 1 and 2 respectively). The HMAC implementation (whose security is under the assumption that  $H$  can be modeled as a random Oracle) is implemented as specified in [1, 2]; in this implementation, using `Digest::HMAC_SHA1`, the hash function  $H$  is the SHA-1.

---

**Algorithm 1** Implementation of the signing function  $y \leftarrow S(k, m)$

---

**Require:** key  $k$  and message  $m$

$k_s \leftarrow H_s(k)$  {Hash of the key.}

$y \leftarrow H(k_s \oplus \text{opad}, H(k_s \oplus \text{ipad}, m))$  {HMAC; `opad` and `ipad` are constants [2].}

---



---

**Algorithm 2** Implementation of the verification function  $V(k, m, s)$

---

**Require:** key  $k$  message  $m$  and signed message  $s$

$k_s \leftarrow H_s(k)$

$y \leftarrow H(k_s \oplus \text{opad}, H(k_s \oplus \text{ipad}, m))$

**if**  $y = s$  **then**

**accept**

**else**

**reject**

**end if**

---

## 2 Encrypted conversation

After the initial DH key exchange a typical protocol would use the shared secret key  $k_0$  and use a symmetric cipher to encrypt any messages between the parties. In the OTR

protocol, however, every message must be encrypted with a different key to provide *perfect forward secrecy* [4]. With perfect forward secrecy, even if Eve was sniffing the channel (storing the exchanged messages) and managed to recover the shared key  $k_i$  she cannot use the key to read (*decrypt*) any of the future messages since they will be encrypted using a different key. It is important for the communicating parties to securely forget the keys<sup>1</sup> after the key exchange using methods presented in [3]. Furthermore, ciphertext integrity is implemented by using a semantically secure cipher ( $E_s, D_s$ ) to encrypt the message and a MAC on the ciphertext. The message exchange between Alice and Bob is shown below:

Server (Alice)		Client (Bob)
1 $x_{ai} \leftarrow_R \mathbb{Z}_p^*$		$x_{bi} \leftarrow_R \mathbb{Z}_p^*$
2 $a_i \leftarrow g^{x_{ai}}$		$b_i \leftarrow g^{x_{bi}}$
3 $c_i \leftarrow E_s(k_{i-1}, m_i)$		
4 $h_a \leftarrow S(k_{i-1}, c_i    a_i)$	$\xrightarrow{c_i, a_i, h_a}$	$V(k_{i-1}, c_i    a_i, h_a)$
5 $V(k_{i-1}, b_i, h_b)$	$\xleftarrow{b_i, h_b}$	$h_b \leftarrow S(k_{i-1}, b_i)$
6		$m_i \leftarrow D_s(k_{i-1}, c_i)$
7 $k_i \leftarrow b_i^{x_{ai}} = g^{x_{bi}x_{ai}}$		$k_i \leftarrow a_i^{x_{bi}} = g^{x_{ai}x_{bi}}$

In the original specifications it is required for the parties to remember keys for more than one step – until the other replies; in this implementation, however an automatic reply with no message is used to avoid the need to store old keys.

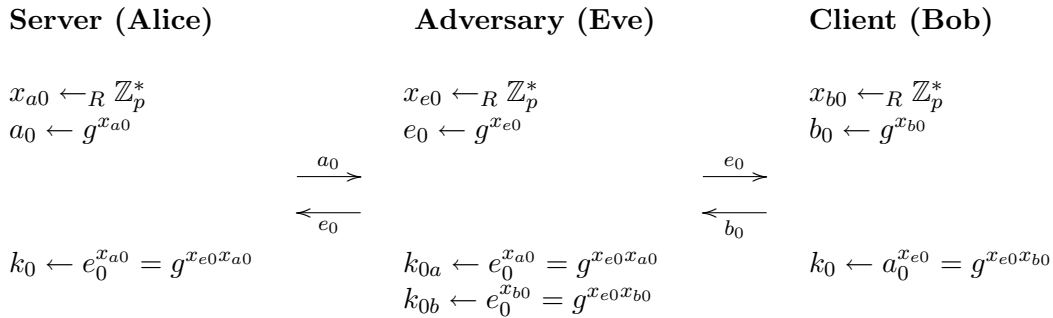
It is important to note that the cipher should be stream cipher, which would allow Eve to modify the ciphertext with meaningful changes in the decrypted plaintext [4] (e.g. XOR with difference of current message and message she wants to inject). As specified in the OTR specifications the cipher used in this implementation is the Advanced Encryption Standard (AES) –Rijndael– in counter (CTR) mode, which is the use of a block cipher as a stream cipher (plaintext is XORed with the keystream).

Finally, Borisov et. al discuss the reviling of the MAC keys  $H_s(k_i)$  after they are used (in this implementation after each message exchange) so that anyone with access to the keys can be a potential suspect in having been part of the conversation.

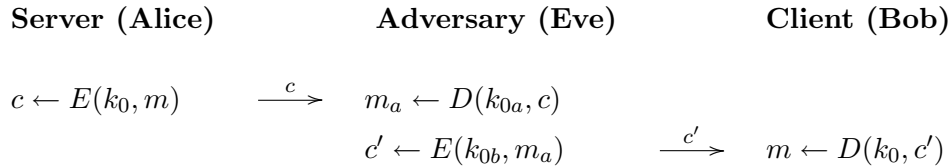
## Appendix I: Man in the Middle Attack on DH Key Exchange

If no authentication method is employed during the key exchange a trivial man in the middle attack can be carried out as shown bellow:

<sup>1</sup>Because this is just a simple proof-of-concept and the methods presented in [3] are quite complex the secure forget was omitted in the implementation.



At the end of the key exchange, Eve has two shared keys, one for the channel with Alice and the other with Bob – for the attack to be carried out successfully it is important for Eve to decrypt any message with the keys she has, encrypt with the other key and forward it to the destination (a denial of service attack would be easily detected otherwise). A message exchange is shown below:



Of course, Eve can now read any of the messages between the Alice and Bob.

## References

- [1] H. Krawczyk, M. Bellare, R. Canetti, Keying Hash Functions for Message Authentication, CRYPTO 1996
- [2] H. Krawczyk, M. Bellare, R. Canetti, RFC 2104, HMAC: Keyed-Hashing for Message Authentication, February 1997
- [3] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, M. Jakobsson, How to Forget a Secret STACS 99
- [4] N. Borisov, I. Goldberg, E. Brewer, Off-the-Record Communication, or, Why Not To Use PGP, Workshop on Privacy in the Electronic Society 2004