Cryptography and Network Security Project Report

Double Ratchet Algorithm

Paper Study and Implementation

Preey Shah	160050008
Abhro Bhuniya	160050017

Contents

1 Introduction 3

2 Overview 3

2.1 KDF Chains 3

2.2 Symmetric-Key Ratchet 3

2.3 Diffie-Hellman Ratchet 4

3 Double Ratchet 5

3.1 Out Of Order Messages 5

4 Implementation 6

5 Security considerations 7

1 Introduction

For our Cryptography and Network Security Course Project, we delve into an encryption algorithm based on key management, the Double Ratchet Algorithm. It can be used as part of a cryptographic protocol to provide end-to-end encryption for instant messaging. We wish to gather insight into the working of the algorithm, its security considerations and try to simulate the algorithm on our machines.

The Double Ratchet algorithm[1] is used by two parties to exchange encrypted messages based on a shared secret key. The parties derive new keys for every Double Ratchet message so that earlier keys cannot be calculated from later ones. The parties also send Diffie-Hellman public values attached to their messages. The results of Diffie-Hellman calculations are mixed into the derived keys so that later keys cannot be calculated from earlier ones.

2 Overview

2.1 KDF Chains

A KDF chain is a core concept in the Double Ratchet algorithm. A **Key Derivation Function** is a cryptographuc function that takes a secret and random KDF key and some input data and returns some output data indistinguishable from random data (provided the key is not known). Thus a KDF is a Pseudo Random Function. If the key is not secret and random, the KDF should still provide a secure cryptographic hash of its key and input data.

In a KDF chain, some of the output of a KDF is used as an output key and some is used to replace the KDF key, which can then be used with another input. A KDF chain has the following properties:

- **Resilience:** The output keys appear random to an adversary without knowledge of the KDF keys. This is true even if the adversary can control the KDF inputs.
- **Forward Security:** Output keys from the past appear random to an adversary who learns the KDF key at some point in time.
- **Break-in recovery:** Future output keys appear random to an adversary who learns the KDF key at some point in time, provided that future inputs have added sufficient entropy.

In a Double Ratchet session between Alice and Bob each party stores a KDF key for three chains: a **root chain**, a **sending chain**, and a **receiving chain** (Alice's sending chain matches Bob's receiving chain, and vice versa).

2.2 Symmetric-Key Ratchet

Every message sent or received is encrypted with a unique message key. The message keys are output keys from the sending and receiving KDF chains. The KDF keys for these chains are called chain keys.

The sending and receiving chains just ensure that each message is encrypted with a unique key that can be deleted after encryption or decryption. Calculating the next chain key and message key from a given chain key is a single **ratchet step** in the symmetric-key ratchet.

2.3 Diffie-Hellman Ratchet

To ensure Break-In Recovery, the Double Ratchet combines the symmetric-key ratchet with a Diffie-Hellman ratchet which updates chain keys based on Diffie-Hellman outputs. As Alice and Bob exchange messages they also exchange new Diffie-Hellman public keys, and the Diffie-Hellman output secrets become the inputs to the root chain. The output keys from the root chain become new KDF keys for the sending and receiving chains.

Each party generates a DH key pair (public key and private key) which becomes their current ratchet key pair. Every message from either party begins with a header which contains the sender's current ratchet public key. When a new ratchet public key is received from the remote party, a **DH ratchet step** is performed which replaces the local party's current ratchet key pair with a new key pair. This results in a "ping-pong" behavior as the parties take turns replacing ratchet key pairs. An adversary who briefly compromises one of the parties might learn the value of a current ratchet private key, but that private key will eventually be replaced with an uncompromised one. At that point, the Diffie-Hellman calculation between ratchet key pairs will define a DH output unknown to the attacker.

1. **Alice Sending Initialization** Alice is initialized with Bob's ratchet public key. Alice's ratchet public key is not yet known to Bob. As part of initialization Alice performs a DH calculation between her ratchet private key and Bob's ratchet public key.

2. **Bob Receiving Initialization** Alice's initial messages advertise her ratchet public key. Once Bob receives one of these messages, Bob performs a DH ratchet step: He calculates the DH output between Alice's ratchet public key and his ratchet private key, which equals Alice's initial DH output.

3. **Ping-Pong step: Bob** Bob then replaces his ratchet key pair and calculates a new DH output. Messages sent by Bob advertise his new public key.

4. **Next step of ping-pong: Alice** Eventually, Alice will receive one of Bob's messages and perform a DH ratchet step: one DH output using Bob's public key and her own public key, one that matches Bob's latest DH output. Then she will replace her ratchet key pair and derive a new DH output, using this new private key and Bob's public key. New messages sent by Alice advertise her new public key.

The last step is repeated by both Bob and Alice whenever either receives a message with a new Public Key from the other.

The DH outputs generated during each DH ratchet step are used to derive new sending and receiving chain keys. Bob uses his first DH output to derive a receiving chain that matches

Alice's sending chain. Bob uses the second DH output to derive a new sending chain which will match Alice's receiving chain. However, instead of taking the chain keys directly from DH outputs, the DH outputs are used as KDF inputs to a root chain, and the KDF outputs from the root chain are used as sending and receiving chain keys. Using a KDF chain here improves resilience and break-in recovery.

3 Double Ratchet

Combining the symmetric-key and DH ratchets gives the Double Ratchet:

- When a message is sent or received, a symmetric-key ratchet step is applied to the sending or receiving chain to derive the message key.
- When a new ratchet public key is received, a DH ratchet step is performed prior to the symmetric-key ratchet to replace the chain keys.

Alice is initialized with Bob's ratchet public key and a shared secret which is the initial root key. As part of initialization Alice generates a new ratchet key pair, and feeds the DH output to the root KDF to calculate a new root key (RK) and sending chain key (CKs).

When Alice sends her first message A1, she applies a symmetric-key ratchet step to her sending chain key, resulting in a new message key. The new chain key is stored, but the message key (after using to encrypt the message) and old chain key can be deleted.

If Alice receives a response B1 from Bob which contains a new ratchet public key. Alice applies a DH ratchet step to derive new receiving and sending chain keys. The first DH output is input to Root KDF with root key RK giving the receiving chain key (CKr). Then she applies a symmetric-key ratchet step to the receiving chain (CKr) to get the message key for the received message. The second DH output is input to Root KDF to get the sending chain key for the new sending chain.

Suppose Alice now sends three messages to Bob, her sending chain will ratchet (symmetric-key) 3 steps deriving 3 message keys for every message in each step. Suppose Alice receives a message with Bob's old ratchet public key, her receiving key will ratchet once.

3.1 Out Of Order Messages

The Double Ratchet handles lost or out-of-order messages by including in each message header:

- N - The message's number in the sending chain (N = 0,1,2,...). On receiving a message, the received N is the number of skipped messages, in the new receiving chain incase a DH ratchet step is triggered or in the current receiving chain.

- PN - The length (number of message keys) in the previous sending chain. On receiving a message, if a DH ratchet step is triggered then the received PN minus the length of the current receiving chain is the number of skipped messages in that receiving chain.

This enables the recipient to advance to the relevant message key while storing skipped message keys in case the skipped messages arrive later.

4 Implementation

- **GENERATE_DH():** Returns a new Diffie-Hellman key pair. The (SK, PK) is given by (x, g^x) where g is a generator and the group is Z_p^* for a prime number p.
- **DH(dh_pair, dh_pub):** Returns the output from the Diffie-Hellman calculation between the private key (x) from the DH key pair dh_pair and the DH public key dh_pub (g^y). It is given by g^{xy}
- **KDF_RK(rk, dh_out):** Returns a pair (32-bit root key, 32-bit chain key) as the output of applying a KDF keyed by a 32-bit root key rk to a Diffie-Hellman output dh_out. HKDF was implemented from the paper *Cryptographic Extraction and Key Derivation: The HKDF Scheme* [2].. The scheme was as below

  $$HKDF(rk, dh\_out, CTX) = K(1)||K(2)||...||K(t)$$

  where

  $$PRK = HMAC(rk, dh\_out)$$
  $$K(1) = HMAC(PRK, CTX||0)$$
  $$K(i+1) = HMAC(PRK, K(i)||CTX||i)$$

  This was as recommended in the paper [1].
- **KDF_CK(ck):** Returns a pair (32-bit chain key, 32-bit message key) as the output of applying a KDF keyed by a 32-byte chain key ck to some constant. We used $HMAC$ for the function as was recommended.
- **ENCRYPT(mk, plaintext, assoc_data):** In our implementation, returns mk XOR plaintext as ciphertext and Ns, PN pair as header in assoc_data. Can change as required.
- **DECRYPT(mk, ciphertext, assoc_data):** In our implementation, returns mk XOR ciphertext as plaintext and updates Nr, PN from values in assoc_data. Can change as required.
- **SEND(msg):** Two versions of send each for Alice and Bob. Ratchet Encrypt for sender is called with the message. Ratchet Decrypt for receiver is then called with the ciphertext and header. This function assumes the same DH key pair as before.
- **UPDATE_SEND(msg):** Two versions of update_send each for Alice and Bob. This function is called to send ciphertexts with a new Public Key. An equivalent of DH step is done where new DH pair is generated for the sender. Ratchet Encrypt for sender is called with the message. Ratchet Decrypt for receiver is then called with the ciphertext and header.

5 Security considerations

We tried to understand Proof of security from the paper on the Signal Messaging Scheme[3], an advanced version of Double Ratchet, but did not understand much. Other Security considerations are required, some (but not all) of them listed below:

- **Secure Deletion:** The Double Ratchet algorithm is designed to provide security against an attacker who records encrypted messages and then compromises the sender or receiver at a later time. This security could be defeated if deleted plaintext or keys could be recovered by an attacker with low-level access to the compromised device.
- **Recovery from Compromise:** The DH ratchet is designed to recover security if one (or both) of the parties gets compromised in a session. Despite this,
  - The attacker can use the compromised keys to impersonate that party.
  - The attacker could substitute her own ratchet keys via continuous active man-in-the-middle attack, to maintain eavesdropping on the compromised session.
  - The attacker could modify a compromised partyâĂŹs Random Number Generator so that future ratchet private keys are predictable.

  If a party suspects its keys or devices have been compromised, it must replace them immediately.
- **Deletion of skipped message keys:** Storing skipped message keys introduces some risks:
  - A malicious sender could induce recipients to store large numbers of skipped message keys, possibly causing denial-of-service due to consuming storage space.
  - The lost messages may have been seen (and recorded) by an attacker, even though they did not reach the recipient. The attacker can compromise the intended recipient at a later time to retrieve the skipped message keys.

  Parties should set reasonable per-session limits on the number of skipped message keys that will be stored. Parties should delete skipped message keys after an appropriate interval.
- **Deferring new ratchet key generation** During each DH ratchet step, as the sending chain is not needed right away, the step for generating sending chain key could be deferred until the party is about to send a new message. This would slightly increase security by shortening the lifetime of ratchet keys, at the cost of some complexity.

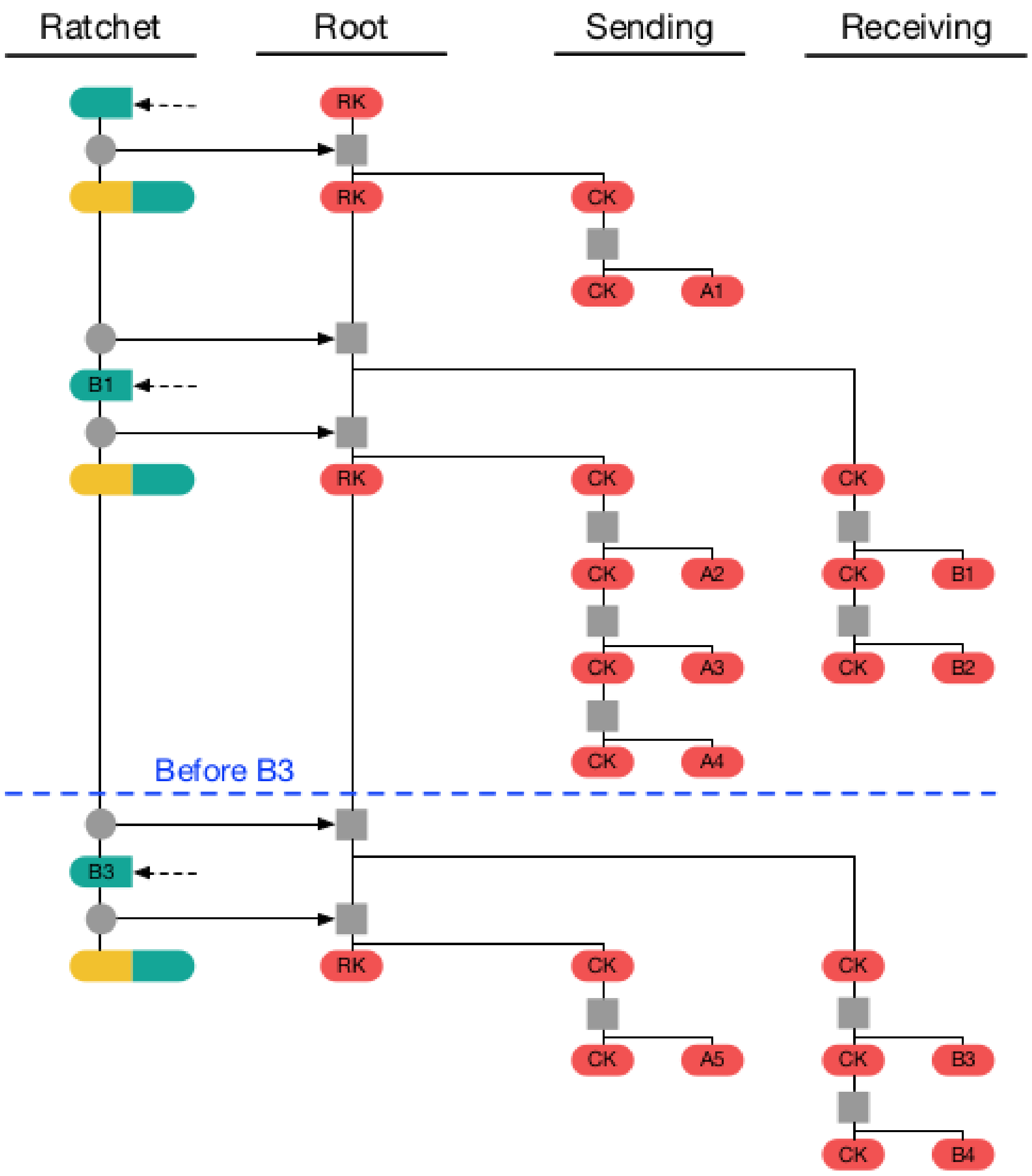

Figure 5.1: Double Ratchet: KDF Chains of Alice

References

[1] Trevor Perrin and Moxie Marlinspike. Double ratchet algorithm. 2016.

[2] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 631–648, 2010.

[3] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 451–466, 2017.