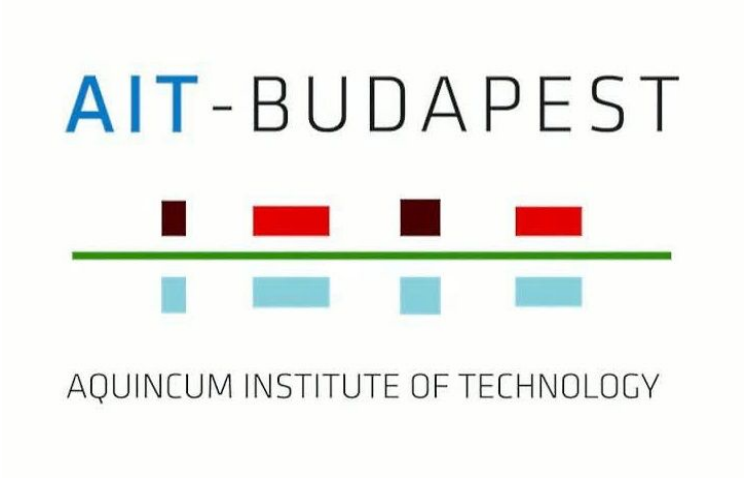


Design Specification

APPLIED CRYPTOGRAPHY

Ankit Sanghi, Mark Rubakh, Winston Wang,
Aurnov Chattopadhyay, Snow Kang

Fall 2019

Supervisor: Dr. Levente Buttyán

Abstract:
An original cryptosystem for file storage, transfer, and queries. Crafted by dedicated engineers after countless hours and tears. A masterpiece for the ages. A series of unbreakable protocols never seen before….

Table of Contents

1 Application Overview

1.1 Functional Requirements

This design document outlines a proposed secure file transfer application. We outline a series of protocols that enables the client and server programs to securely communicate. Our proof of concept server may support multiple user accounts, but only one user will interact with the server at a time via a command line interface.

Our applications supports the following commands to modify a file system.

MKD – creating a folder on the server	**LST** – listing the content of a folder on the server
RMD – removing a folder from the server	**UPL** – uploading a file to the server
GWD – get the current folder name on the server	**DNL** – downloading a file from the server
CWD – changing the current folder on the server	**RMF** – removing a file from a folder on the server

In this document, we identify a number of attacker capabilities, goals, and trust assumptions to derive core security requirements. Based on these requirements, we establish and outline protocols for server verification and user login to initiate a session and command request-execution-confirmation sequences.

Our protocols are designed using cryptographic encryption schemes to support confidentiality, authenticity, integrity, and replay protection on our application.

2) Attacker Models & Trust Assumptions

2.1 Attacker Capabilities

We assume attackers are capable of
[1] eavesdropping on all messages between the server and client,
[2] modifying or intercepting messages sent between the server and client, and
[3] sending original messages to the server or client.

2.2 Goals

Attacker motivations may include
[1] impersonating a valid user to modify, read, or download components of the file system,
[2] identifying characteristics about the file system and requests made by listening on interactions, and
[3] blocking the execution of client actions.

2.3 Trust Assumptions

We assume that
[1] the attacker does not have access to a valid client password or an account,
[2] the attacker does not have and will not gain access to the out-of-band public-private key pair,
[3] the attacker is unable to break cryptographic primitives,
[4] we can trust the server after it has been authenticated by cryptographic means, and
[5] we can trust the client after they have been authenticated by login with their password.

3 Security Requirements

3.1 Core Requirements Based on our stated attack models, we derive the following core security requirements for our protocol:

- **Confidentiality**: An attacker eavesdropping on client messages to the server should not be able to identify the type of command or file name used.
- **Integrity Protection:** The attacker should not be able to intercept messages between the client and server or modify specific parts of the message to produce a known outcome or have a known impact on the file system without being detected/exposed.
- **Authenticity:** Any commands reaching the server must be authenticated as coming from the client. Any command execution responses reaching the client must be authenticated as coming from the server. Authentication is critical because our attacker may aim to intercept and impersonate the server or the client in their interactions.
- **Replay Protection:** Neither the client nor the server should accept previous command messages or execution messages that the attacker may have overheard and replayed. This includes protection of replayed messages that the attacker has modified.

3.2 Non-requirements Our security requirements exclude the following:

- **Perfect Forward Secrecy:** Even if the attacker eavesdrops on a series of interactions and later identifies the shared secret key for an interaction, they will not be able to decrypt the series of interactions and learn about the file system. Our trust assumption is that the attacker will be unable to identify a shared secret key.
- **Non-Repudiation** (Of Command Origin or Reception): Not satisfied in general.

4 Protocols

4.1 Keys

Long-term:

[1] Types

- Certified public-private key pair owned by the server
- Passwords between clients and server

[2] Availability

- The passwords are considered permanent
- The public-private key pair is renewed if either key is suspected to be compromised or the server wants to update certification information

[3] Storage

- All passwords are stored in files protected by a passphrase
- We assume that the server/client memorizes the relevant passphrases
- The server stores the key pair in a file named “keypair.pem”
- The server stores the hash of the passwords in a file named “passwords.txt”
- The client stores the public key in a file named “pubkey.pem”
- The client stores the password in a file named “password.txt”

Short-term:

[1] Types

- Session keys generated by the client

[2] Availability

- Keys are only used for the duration of each session

- After the session we delete the file “sessionkey.txt”

[3] Storage

- The server stores the client session key in a file named “sessionkey.txt”
- After the session is over, the file is deleted

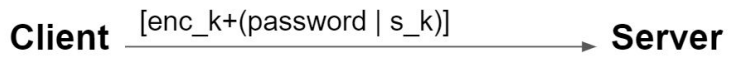
4.2 Protocols

Session Login Protocol

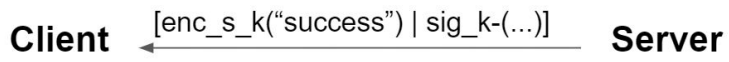
Notation:

- Private key: k-
- Public key: k+
- Session key: s_k

Description:



Server
decrypted_message = dec_k-[received message]
if enc_k+(password) in passwords:
    sends [enc_s_k(“success”) | sig_k-(...)] after 10 seconds from receiving message
else:
    sends [enc_s_k(“failure”) |sig_k-(...)] after 10 seconds from receiving message



Client
if signature verifies correctly and login “success”:
    proceed to session command protocol
else:
    quit or login again

Notes & Assumptions:

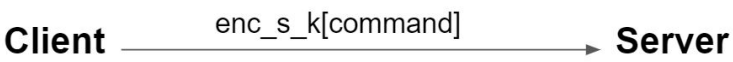
- s_k is generated by the client
- At any moment, there can only be one session in progress

Session Command Protocol

Notation:

- Session key: s_k

Description:



Notes & Assumptions:

- For the command and response specifications, see section 5.0

4.3 Cryptographic algorithms used and their parameters

Session Login Protocol

Asymmetric Encryption/Decryption:

- Cryptosystem: RSA
- Keys: Generated by pycryptodome (2048 bits)
- Digital Signature: PKCS #1 with SHA256

Symmetric Encryption/Decryption:

- Keys: Randomly generated from cryptographic PRNG (128 bits)
- Block cipher: AES
- Mode of Operation: Counter Mode (CTR)
- Counter object: nonce (64 bits) + counter (64 bits)
- Generation of Nonce: Randomly generated from cryptographic PRNG (64 bits)

Session Command Protocol

Symmetric Key: AES Cipher

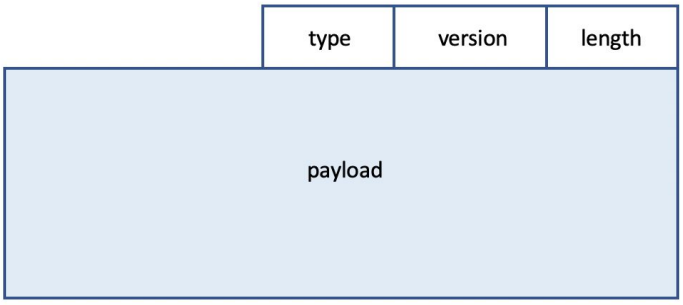
- Key Size: 128 bits
- Mode of Operation: Counter Mode (CTR)
- Counter object: nonce (64 bits) + counter (64 bits)
- Generation of Nonce: Randomly generated from cryptographic PRNG of size 64 bits

4.4 Exact format of each protocol message

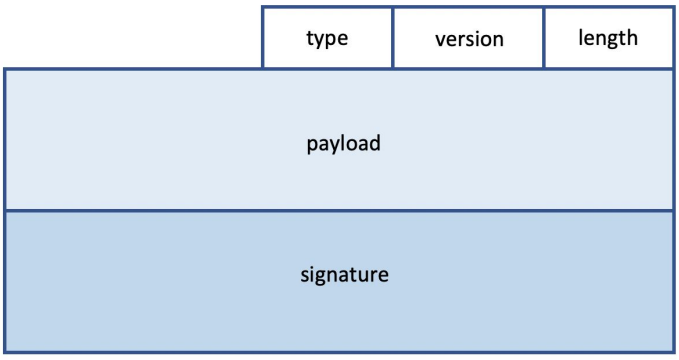
The initial version is 1.0.

[1] Session login protocol (type == 0)

- Client messages:

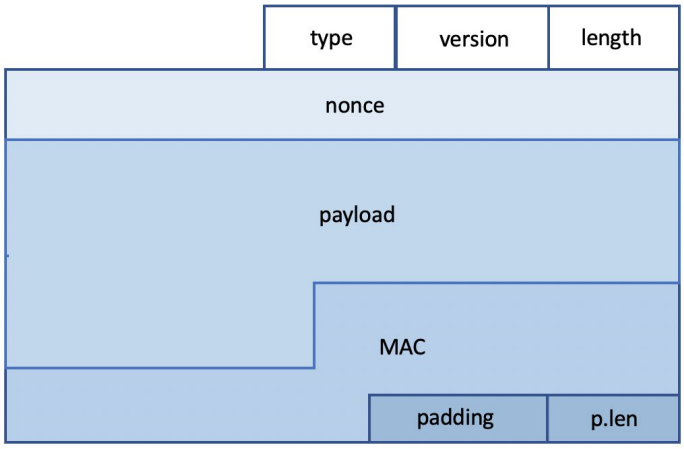


- Server messages:
    - signature uses the server public key



[2] Session command protocol (type == 1)

- All messages:
    - MAC uses the session key



4.5 Justification for security requirements

Session Login Protocol:

- **Server authentication, integrity protection, and non-repudiation of origin** is built into the protocol as the server produces a signature using its private key. This prevents server impersonation.
- **Client authentication** is built into the protocol, since no third party has access to the password and the out-of-band public key.
- **Client integrity protection** is built into the protocol. If s_k is modified, the client will not be able to decrypt the server message properly, and the current session will be terminated. Since RSA is used to encrypt the client's login message, the attacker has no good way of modifying the password without changing s_k as well.
- **Replay attack protection** is provided by the generation of a fresh s_k, which does not allow repetition of the client server messages.
- **Confidentiality** of the "success" or "failure" response in the protocol is baked into the encryption of the response.
- **Brute force attack** prevention is offered by a wait time of 10 seconds. The specific waiting time may be subject to change.

- **Oracle style attacks** relying on the server to verify message validity is avoided by the verification of the MAC in the message format.

Session Command Protocol:

- **Confidentiality** of the messages sent in the protocol is baked into the encryption of the response message.
- **Integrity** is provided by the MAC in the message format.
- **Authenticity** is given after a successful session login protocol.
- **Brute force attack** prevention is offered by the key space of 2^{128}.

5 Command Specifications:

5.1 Command Descriptions and Result Messages to User:

MKD –

Usage: MKD -n <name_of_folder>
Success message: "The folder ${folder_name} was created"
Error message: "There was an error in creating the folder."

RMD –

Usage: RMD -n <name_of_folder>
Success message: "The folder ${folder_name} was removed"
Error message: "There was an error in removing a folder."

GWD –

Usage: GWD
Success message: "The current folder is ${current_folder_name}"
Error message: "There was an error in getting the name of the current folder."

CWD –

Usage: CWD -p <path_to_folder>
Success message: "The current folder is now ${current_folder_name}"
Error message: "That path is invalid or that folder could not be found."

LST –

Usage: LST
Success message: a list of all the items in the current folder, one on each line
Error message: "There was an error in listing out the contents of the folder ${current_folder_name}"

UPL –

Usage: UPL -f <filename>
Success message: "The file ${filename} was uploaded to the folder ${current_folder}"
Error message: "There was an error in uploading the file."

DNL –

Usage: DNL -f <filename> -d <destination_path>
Success message: "The file ${filename} was downloaded from the folder ${current_folder}"
Error message: "There was an error in downloading the file."

RMF –

Usage: RMF -f <filename>
Success message: "the file ${filename} was removed from the folder ${current_folder}"
Error message: "There was an error in removing the file."

5.2 Response codes

- In the session command protocol, the server sends a code as the response
- If the server executes a command successfully, it sends a success code (200) concatenated with any data relevant to the command.
    - For GWD, server sends the name of the current folder.
    - For LST, server sends a list of items in the current folder.
    - For DNL, server sends the requested file from the current folder.
- The server fails to execute a command, it sends a failure code (500)
- The client listens for the response code:
    - If it is 200, then it sends the success message.
    - Otherwise, it sends the error message.