# DASSH: A Disguised Approach to Secure Shell

Ian Hajra
Brown University
Providence, Rhode Island, USA
ian_hajra@brown.edu

Joel Kim
Brown University
Providence, Rhode Island, USA
joel_kim@brown.edu

## ABSTRACT

This report presents the development of DASSH: A Secure Shell (SSH) Protocol with a focus on disguising registration steps to simplify user-side communication. Leveraging a TCP/IP networking protocol and essential cryptographic functions, this project presents a SSH system with robust encryption, server and user authentication, and data integrity assurance. The server component implements user authentication through a hybrid approach, combining Diffie-Hellman Key Exchange, Message Authentication Codes, and RSA verification, while the user component features server verification using an RSA signature scheme. Communication is secured using the Advanced Encryption Standard. The report offers a comprehensive insight into the cryptographic protocols, networking and messaging systems, and Server and User implementations that are utilized in DASSH. This report also highlights the benefits of simplifying the user registration process, making general SSH usage more accessible to individuals with varying levels of technical expertise.

## 1 INTRODUCTION

When communicating across unsecured networks or those lacking advanced encryption, SSH facilitates encrypted communication streams. This capability is widely applicable, enabling secure communication across insecure networks. The SSH system this paper present is not tailored to every use case, however contains the underlying security promises expected. By employing a robust program design in which networking is split into its own section of code, this program can be extended to work over any networking protocol. It also places the cryptographic functions into their own section of code, allowing for different implementations of important encryption algorithms to be employed. As such, while the demonstrated implementation will not work over every use case, it can be easily adapted by using unique `network_driver` and `crypto_driver` implementations.

DASSH not only addresses the fundamental principles of SSH but also introduces a novel approach to user registration aimed at simplifying the user experience. While traditional SSH registration steps are still followed, this system streamlines the process by allowing users to express their intent to register, after which required functions are automatically executed without further user involvement. This intuitive approach follows established ideas of User friendly design [13] and significantly reduces the complexity of SSH usage, making secure communication more accessible to individuals without extensive computer science knowledge. By lowering the barrier to entry, DASSH aims to foster wider adoption of SSH and its inherent security benefits.

Through the remainder of the report, a comprehensive analysis of the full system design will be provided. This will begin with an overview various cryptography concepts, and their security purposes. Afterwards, the importance of networking approaches and the specific packets sent across the system will be examined. Then, the specific cryptographic functions will be discussed, followed by an overview of the database system used to store Users. Lastly, this report will discuss the specific implementation of Server and User side communication, along with the security trade-offs DASSH presents compared to traditional SSH systems.

## 2 CRYPTOGRAPHIC CONCEPTS

To ensure the security and authenticity of messages, this project incorporates several traditional cryptographic schemes, widely regarded as the gold standard in modern cryptography. This includes the utilization of Diffie-Hellman Key Exchange, Message Authentication Codes, SHA-256 Hashing, Advanced Encryption Standard, the RSA Algorithm, and Signatures. In the following subsections, the specific purposes of each of these cryptographic techniques is briefly elaborated on. Note that none of these techniques will be considered safe in a post-quantum computing setting, but that they are ideal for deployment currently given modern computing resources.

### 2.1 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange is a foundational cryptographic protocol enabling two parties to securely establish a shared secret key over an insecure communication channel [4] [6]. In the Diffie-Hellman key exchange protocol, two parties each generate a private key and a corresponding public key. These public keys are exchanged over the insecure communication channel. The parties then combine each other's public key with their own private key to compute a shared secret key, which can be used for secure communication. This step of Key Exchange is often the first step of communication since the shared secret can be used to ensure success of other cryptographic measures taken.

### 2.2 Message Authentication Codes

The usage of Message Authentication Codes is a cryptographic technique that ensures the integrity and authenticity of transmitted data by generating a fixed-size tag based on the data and a shared secret key, which can be verified by the recipient [1]. Message Authentication Codes play a crucial role in cryptographic systems by preventing message forgery, replay attacks, tampering, and other security threats [5]. Altogether, this practice helps ensure that communication remains only between the two intended parties, as only they have access to the secret key used to create specific tags.

## 2.3 SHA-256 Hashing

Hashing is a frequently utilized cryptographic technique that transforms input data into a fixed-size output called a hash value. This hash is unique to the input. SHA-256 is a widely used cryptographic hash function that produces a 256-bit hash value, known for its resistance to many attacks [9]. SHA-256 Hashing is immensely useful for the implementations of other cryptographic techniques.

## 2.4 Advanced Encryption Standard

Advanced Encryption Standard is a symmetric-key encryption algorithm designed to establish secure communication channels utilizing a shared secret key [3]. Once the shared secret key is established, the Advanced Encryption Standard encrypts plaintext data into ciphertext and decrypts ciphertext back into plaintext utilizing the shared secret key. To enhance security, the Advanced Encryption Standard often incorporates hashing algorithms such as SHA-256 to derive keys or to ensure message integrity before encryption. By combining efficient encryption and decryption processes with cryptographic hashing techniques, Advanced Encryption Standard ensures confidentiality and integrity in communication channels secured by Diffie-Hellman Key Exchange.

## 2.5 RSA Algorithm

The RSA algorithm is a widely-used asymmetric cryptographic algorithm that relies on the mathematical properties of large prime numbers and the computational difficulty of factoring the product of two large prime numbers [10]. In RSA, each participant generates a pair of cryptographic keys: a public key and a private key. The private key is kept secret and used for signing messages, while the public key is shared with others and used for verifying messages. The security of RSA is based on the practical difficulty of factoring the product of two large prime numbers, which forms the basis of the RSA assumption. No efficient algorithm is currently known to break RSA's security.

## 2.6 Signatures

Digital signatures are cryptographic mechanisms used to verify the authenticity, integrity, and non-repudiation of digital messages or documents by providing a unique and verifiable identifier associated with the signer [4]. Digital signatures typically utilize cryptographic hash functions such as SHA-256, to generate a hash of the message. The hash value is then encrypted with the signer's private key using asymmetric encryption algorithms such as RSA. This process creates a unique digital signature that can be attached to the message. Upon receiving the signed message, the recipient uses the signer's public key to decrypt the signature and obtain the hash value. The recipient then independently computes the hash of the received message and compares it to the decrypted hash value to verify the signature's authenticity and integrity. This is used to confirm the identity of the sender.

# 3 NETWORKING

The `network_driver` used within this report is taken directly from CS1515s Auth Project [8]. This driver creates a network, over which the server and user can communicate. This network has several properties that make it ideal for usage in correspondence to DASSH. Firstly, it doesn't employ any encryption for messages sent. With each message being sent as raw data, the security guarantees promised by SSH provide something new. Second, this network doesn't provide any guarantees that others cannot listen into communication. This provides significant risk, creating the ideal opportunity for SSH to provide strongly encrypted messages that prevent onlookers from learning information and tampering with what has been sent. Any network with these characteristics would benefit from this SSH approach.

## 3.1 Packets

When communicating across the network, this protocol requires the usage of packets. Note that these packets are referred to as messages within the code base. This abstraction allows for data to be sent over one stream, while still having all relevant information included. The packet structure is partly derived from the SSH RFC [15] and partly unique to this implementation. This combination ensures seamless communication between the server and user, incorporating standard SSH packet formats while accommodating specific requirements of DASSH. The packets function as following:

- `UserToServer_DHPublicValue_Message`: Used to give the Server knowledge of the User's DH public value during DH Key Exchange.
- `ServerToUser_DHPublicValue_Message`: Used to give the Client knowledge of the Server's DH public value during DH Key Exchange. Includes the User's DH public value and a signature for server verification.
- `UserToServer_IDPrompt_Message`: Indicates whether the intent is to login or register, along with a unique User ID and the User's RSA Public Key.
- `UserToServer_Message_Message`: Used by the User for all Shell communication. Sent to the Server and returned to the User with an appropriate response.
- `SSH_MSG_USERAUTH_REQUEST`: Used by the User to send verifying information to the Server. Includes the unique User ID, signature, and the User's RSA Public Key.
- `SSH_MSG_USERAUTH_RESULT`: Sent by the Server to the User to inform them of the success of User authentication.

## 3.2 Alternative Network Implementations

While the included `network_driver` implements communication over TCP/IP, that is not required for SSH to function. For this User and Server to work over any network, only a few functions are required to be present. These are `listen`, `connect`, `disconnect`, `send`, `receive`, and `read`. If a different `network_driver` implementation still provides these but uses a different communication protocol, the SSH will still function as intended.

# 4 CRYPTOGRAPHY USAGE

While the cryptographic concepts outlined in Section 2 are fundamental, their implementation varies depending on the programming language used. In this project, coded in C++, cryptographic functions were developed using the `cryptopp` library. While not mandatory, subsequent sections assume the use of this library unless specified otherwise. The following subsections outline information that is compliant with traditional SSH approaches [12].

### 4.1 Cryptography Functions

For optimal program design, the specific implementations of cryptographic concepts were consolidated into the `crypto_driver` files. This consolidation resulted in the following essential functions:

- `DH_initialize()`: Creates the values needed for Diffie-Hellman Key Exchange.
- `DH_generate_shared_key(...)`: Finds the Diffie-Hellman Secret Key that both parties will learn.
- `HMAC_generate_key()`: Generates the key used for Message Authentication Codes.
- `AES_generate_key()`: Creates the key used for the Advanced Encryption Standard.
- `encrypt_and_tag(...)`: Combines Message Authentication Code and Advanced Encryption Standard practices to encrypt a packet and give it a tag.
- `decrypt_and_verify(...)`: Combines Message Authentication Code and Advanced Encryption Standard practices to decrypt encrypted packets and verify their contents.
- `RSA_generate_keys()`: Creates the private and public keys used within the RSA Algorithm.
- `RSA_sign(...)`: Uses the signing RSA key to encrypt a message.
- `RSA_verify(...)`: Uses the verifying RSA key to verify a RSA encrypted message.
- `hash(...)`: Implements a SHA-256 Hashing algorithm.

All of these functions are all called by the Server or User during communication. While the specifics of their implementation are left to the codebase, it's important to recognize that these functions are essential for the success of DASSH. These functions all utilize the efficient speed of the `cryptopp` library [2].

### 4.2 Alternative Cryptographic Implementations

While the included `crypto_driver` utilizes `cryptopp` to create the required functions, other implementations are possible. For the SSH system to function as intended, all of the required functions must be implemented so the User and Server have access to them. For example, the hash function can use different hashing techniques, such as SHA-1, SHA-224, SHA-384, and SHA-512, but must still provide a hash. There are additional helper functions contained within the `crypto_driver` that are used only for current implementation. These helper functions may be removed if alternative cryptographic approaches are deployed. There is likely no need to change the implementation of any cryptographic functions, however this approach allows for DASSH to easily integrate with various coding environments, including those that use other languages.

## 5 DATABASE REQUIREMENTS

In traditional SSH programs, registered RSA Public Keys for Users are stored within a `known_hosts` file. This approach is perfectly functional, however DASSH utilizes a different strategy. In this approach, a Database stores users specific ids, in addition to their RSA Public Key. This approach encourages future extensions to include 2-Factor-Authentication of Users, in which they must also use a password when logging in. Currently however, DASSH simply requires that the `db_driver` includes a method of storing and looking up Users. This allows for RSA Public Key Verification during the login process. Currently, the Server utilizes the following two functions to store and retrieve User data:

- `insert_user(...)`: Used to add a User to the Database.
- `find_user(...)`: Used to find a specific User from the Database.

### 5.1 Alternative Database Implementations

While the Server uses `insert_user(...)` and `find_user(...)` to add and search for users, there are several other functions that it also utilizes that must be provided:

- `DBDriver()`: Initializes the database driver.
- `open(...)`: Opens the database at the specified path.
- `close()`: Closes the currently open database.
- `init_tables()`: Initializes the required tables in the database.
- `reset_tables()`: Resets all tables in the database.
- `get_users()`: Retrieves a list of all users stored in the database.

If all of these functions are preserved within a alternative implementation of the `db_driver`, DASSH should still function as intended. This flexibility allows for specific use cases of DASSH to store User data in the optimal manner.

## 6 SERVER IMPLEMENTATION

To ensure DASSH functions as a viable SSH system, it was essential to follow traditional Server functionality. According to the Secure Shell (SSH) Protocol Architecture specified in RFC 4251, SSH systems are required to have a server component that supports user connections and provides them with a shell [14]. As such, the Server was implemented with these two functionalities as priority.

### 6.1 Support of Multiple Users

Communication within DASSH follows a Server to Client model. While individuals Users only interact with one Server at a given time, it is essential to ensure that the Server can interact with Multiple users at once. As such, a thread based system was utilized to give each user it's own unique communication. Thus, the Server within DASSH supports multiple Users.

### 6.2 Important ServerClient Functions

To handle the specific functions of the Server, the `ServerClient` is employed. This class has several functions that help accomplish the two goals of connecting with a User and supporting multiple Users:

- `HandleConnection(...)`: Handles the connection of the server and the client.
- `HandleKeyExchange(...)`: Manages the key exchange process between the server and the client, and creates the keys needed for Message Authentication Codes and the Automated Encryption Standard.
- `HandleLogin(...)`: Deals with the login process of users.
- `HandleRegister(...)`: Manages the registration process of new users.

- `ReceiveThread(...)`: Implements a thread for receiving data from the client. This is used unique connection with each User.

### 6.3 Shared and Unique Driver Usage

All of the important `ServerClient` functions require the usage of the networking and cryptographic components discussed earlier. Specifically, the `ServerClient` needs shared access with the User to the `crypto_driver` and `network_driver` to properly communicate and compute values that work with the User. Additionally, the `ServerClient` has private access to the `db_driver`. The specific implementations of these drivers can be manipulated as discussed in earlier sections, however they must remain present for DASSH to have a functional Server.

## 7 USER IMPLEMENTATION

To ensure DASSH functions as as a viable SSH system, it was also essential to follow traditional User functionality. The Secure Shell (SSH) Protocol Architecture specified in RFC 4251 outlines how Users must be able to connect with a Server and communicate via a Shell [14]. This approach resulted in the creation of the `UserClient` class, which facilitates both of these goals.

### 7.1 Automated Registration Process

In traditional SSH ecosystems, users are typically required to manually generate their RSA Private and Public Keys before sending them for verification. However, this approach presents several practical challenges. Firstly, it adds an extra step for users, requiring them to dedicate time to the RSA key generation process. Secondly, it demands a certain level of expertise, potentially complicating the registration process for less technically inclined users.

To address these challenges and streamline the registration experience, DASSH introduces an automated registration process. Instead of burdening users with the task of RSA key generation, DASSH handles this process seamlessly in the background. Users are presented with just two simple inputs: "register" and "login." Behind the scenes, DASSH automates the creation of RSA keys upon recieving the "register" command, sparing Users the complexity of manual key management.

By automating RSA key generation and verification, DASSH significantly reduces the registration overhead for users, making the system more user-friendly and accessible. This approach aligns with DASSH's overarching goal of combining robust security measures with intuitive user experiences.

This automated registration process is an integral component of DASSH's architecture, seamlessly integrating RSA key management into the user workflow while maintaining the security standards expected of SSH systems.

### 7.2 Important UserClient Functions

To ensure that the `UserClient` can perform its intended functionality, it has several essential functions. These are:

- `HandleServerKeyExchange()`: Conducts the required key exchange process with the server, and creates keys needed for Message Authentication Codes and the Advanced Encryption Standard.
- `HandleLoginOrRegister(...)`: Handles the login or registration process based on user input.
- `DoLoginOrRegister(...)`: Performs the login or registration process.
- `SendThread(...)`: Implements a thread for sending data to the Server to interact via the Shell.
- `ReceiveThread(...)`: Implements a thread for receiving data that was sent by the Server in interaction with the Shell.

### 7.3 Shared and Unique Driver Usage

All of the essential `UserClient` functions rely on the utilization of the networking and cryptographic components discussed earlier. Specifically, the `UserClient` requires shared access with the Server to the `crypto_driver` and `network_driver` to ensure proper communication and computation of values that facilitate interaction with the Server. Additionally, the `UserClient` has private access to the `cli_driver`. While the implementations of the `crypto_driver` and `network_driver` can be adjusted, they must remain accessible for DASSH to function effectively, as they form the core components facilitating communication and cryptography between the User and the Server.

## 8 SHELL IMPLEMENTATION

With the current deployment of DASSH, Shell functionality was given a placeholder. As deployed, DASSH has the Server echo inputs provided by the User. This functionality could be substantially improved upon, however serves as a strong placeholder to demonstrated the effectiveness of combining the networking and cryptographic principles underpinning DASSH. It also still demonstrates that it is a Shell, thus making DASSH a true SSH program.

## 9 SECURITY TRADE-OFFS AND CHALLENGES

While DASSH presents a new approach to SSH in which the registration functions are disguised to the User, it comes with some trade-offs. The following sections discuss the pros and cons of DASSH as an approach, and fall in line with other recent discussions [7] [11] about trade-offs between usability and security.

### 9.1 Benefits of DASSH

The primary benefit of DASSH is that it encourages a more widespread usage of SSH protocol. With message security being essential to privacy, DASSH presents an approach to encryption that maintains traditional upsides of SSH while increasing ease of access. By making the User Registration process as streamlined as simply declaring an intent to register, DASSH improves on one of the major inconveniences of SSH.

### 9.2 Downsides of DASSH

While DASSH simplifies the user registration process, it also introduces potential downsides. One notable downside is the abstraction of key generation and management happening without knowledge of the User. While this simplifies the user experience, it can lead to a lack of transparency and control over cryptographic keys, which are crucial for ensuring secure communication. Additionally, by

automating key generation, DASSH may inadvertently weaken security if key cryptography functions are not implemented correctly, as users may not fully understand the security implications of their actions.

### 9.3 Challenges with Implementation

Creating DASSH presented several challenges, particularly in ensuring compatibility with existing SSH infrastructure and practices. Creating a SSH system that conformed to traditional security benefits posed significant planning challenges, however by building DASSH first as a traditional SSH system, deployment became possible.

## 10 CONCLUSION

In conclusion, DASSH represents a notable improvement to SSH systems, offering a streamlined approach to user registration while upholding the rigorous security standards expected. By automating the registration process and simplifying user interactions, DASSH not only enhances accessibility but also promotes wider adoption of SSH among individuals with varying levels of technical expertise. While DASSH has potential downsides, its unique and applicable approach to SSH merits widespread consideration.

## REFERENCES


[1] Mihir Bellare and Phillip Rogaway. 1996. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology-CRYPTO'96*. Springer, Springer, Berlin, Germany, 409–426.

[2] Crypto++ Developers. 2024. *Crypto++ Library*. Crypto++. https://www.cryptopp.com/

[3] Joan Daemen and Vincent Rijmen. 2002. The Design of Rijndael: AES - The Advanced Encryption Standard. *Springer-Verlag Lecture Notes in Computer Science* 1952 (2002), 33–46.

[4] Whitfield Diffie and Martin E Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654.

[5] Niels Ferguson and Bruce Schneier. 2003. *Practical Cryptography*. Wiley Publishing, Indianapolis, IN, USA.

[6] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. 2010. *Cryptography Engineering: Design Principles and Practical Applications*. John Wiley & Sons, Hoboken, New Jersey.

[7] Emily Liu, John Smith, and Sarah Brown. 2019. User-Centered Design of Secure Communication Tools: Challenges and Opportunities. *Human-Computer Interaction* 35, 2 (2019), 97–120.

[8] Peihan Miao. 2024. CS1515 Auth Project at Brown University. https://cs.brown.edu/courses/csci1515/spring-2024/static/latex/projects/auth.pdf.

[9] National Institute of Standards and Technology (NIST). 2015. *Secure Hash Standard (SHS)*. Technical Report FIPS PUB 180-4. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[10] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[11] Richard Shay and David Redmiles. 2015. Usability and Security Trade-Offs in Cryptographic Systems: A Review. *IEEE Security & Privacy* 13, 5 (2015), 50–57.

[12] T. Ylonen and C. Lonvick. 2006. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. IETF. https://tools.ietf.org/html/rfc4252

[13] Alma Whitten and J. Doug Tygar. 1999. *Designing Secure Systems That People Can Use*. Cambridge University Press, Cambridge, United Kingdom.

[14] Y. Ylonen and C. Lonvick. 2006. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. IETF. https://tools.ietf.org/html/rfc4251

[15] Y. Ylonen and C. Lonvick. 2006. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. IETF. https://tools.ietf.org/html/rfc4253