Cryptography: How to Build an Intuitive Cryptographic Library without Sacrificing Power

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Daniel Evan Farmer

Spring, 2025

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman, Department of Computer Science

Cryptography: How to Build an Intuitive Cryptographic Library without Sacrificing Power

CS4991 Capstone Report, 2025

Daniel Farmer
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
gas2me@virginia.edu

ABSTRACT

Many cryptographic libraries exist in computer programming, but most are either easy to use or powerful, never both. To fix this, I built a robust, cryptographic library in the C programming language that prioritizes an intuitive user experience, while still providing extensive cryptographic operations. This library modeled its semantics and style after Libsodium, a modern crypto library that is very intuitive but offers limited operations to its users. The library was built on top of Libgcrypt, a powerful toolkit with a high learning curve, to make use of its low-level encryption/decryption implementations. This resulted in a comprehensive library that allows the user to implement all necessary cryptographic operations, plus some more niche ones, in just a few lines. Future work on this library could be done to increase the support of different data types used to store encrypted data. Currently the library accepts and stores data as strings, but a second phase would allow the user to be able to encrypt/decrypt all popular data types.

1. INTRODUCTION

So why do we need cryptography, anyway? Cryptography is the practice of hiding or coding information so that only authorized actors can read it. This is very important in the digital age, as many computer programs contain highly sensitive information like passwords, bank account numbers and social security numbers, among other things. As a result, computer programmers use cryptography to keep this information safe from malicious actors.

Cryptographic libraries are toolkits of programming methods used to streamline cryptographic operations. Many of these operations require certain algorithms to be applied to information that needs to be hidden, and rather than implementing that algorithm oneself, users can make use of these libraries' methods that already contain the implementation. The problem lies in the fact that many of these libraries have a very steep learning curve. The libraries built to address this problem are inflexible and not powerful. My project addresses these issues by constructing a powerful cryptographic library that is both intuitive and powerful.

2. RELATED WORKS

Our work is based on Libgcrypt's (2) implementations of different crypto algorithms. Libgcrypt's documentation contains all its methods and how to use them. Immediately we can see several limitations with these docs. First, there is no definition of an s-expression (an important data type in this library) and only a couple of real examples of their use. Second, throughout the paper, most methods do not include examples, and each category usually only has one code example. Essentially this document is an explanation of

each method in the library but provides no documentation on how to use them, with only limited examples showing how to apply them to real code.

Whitten and Tygar discuss the useability of PGP 5.0, one of the most widely used encryption programs available. The authors conclude that: "The analysis found a number of user interface design flaws that may contribute to security failures…" [1]. They cite a study claiming that, given 90 minutes to accomplish basic encryption protocols, most users cannot figure it out. Useability is a weakness of PGP, and this weakness results in security failures and incorrect cryptographic practices.

3. PROJECT DESIGN

This section dives into the construction of the library. This will introduce what requirements the library must meet as well as its actual features. Additionally, it will go over the challenges and solutions presented when building the library.

3.1. Review of System Architecture

The library was built in the C programming language. The goal is for the user to include this file in their C code and immediately have access to every cryptographic operation they may need, while being able to easily understand how to use them.

The project was built on top of Libgcrypt because it contains the implementations of many crypto algorithms that allow for important cryptographic operations. The library was built utilizing Libgcrypt for these specific mathematic operations by importing it in the header.

To promote usability and a short learning curve, I modeled the library after Libsodium, an easy-to-use modern cryptography toolset. I not only mirrored its semantics, but also its method's arguments, allowing the user to store variables as simple strings.

3.2. Requirements

The library was required to provide necessary cryptographic functions for securing data at rest and in transit. Furthermore, the library was required to eliminate any memory leaks, and was run with a popular tool, Valgrind, to accomplish this. The library also needed to be interchangeable with other cryptography libraries, which means encrypting with our library and decrypting with a common toolkit like Libgcrypt, for example. This interchangeability proves that the methods we built accomplish our crypto protocols successfully.

3.2.1. Client Needs

There are some key functions that every crypto library should have to secure data at rest and in transit: asymmetric encryption; symmetric encryption; hashing; and password-based key derivation. These four concepts and their related methods should allow a user to accomplish almost every operation they can think of. This will allow the user to generate keys, send encrypted messages back and force, store their passwords, and much more. The library also needs to be customizable, allowing the user to choose among many different algorithms. Some algorithms are faster on certain operations; some are FIPS compliant (the Federal Information Processing Standard); and some are more secure.

3.2.2. System Limitations

Our library is limited to the cryptographic algorithms that Libgcrypt supports. Libgcrypt supports many algorithms but lacks options for elliptic curves. Our library is essentially only as powerful as Libgcrypt and can only do operations it supports.

3.3. Specifications

The methods this library was built to include are:

- Symmetric encryption and decryption
    - Symmetric key generation
    - Authenticated tag generation
- Asymmetric encryption and decryption
    - Public private key generation
    - Diffie Hellman key exchange
    - Authenticated Encryption
    - Anonymous encryption with ephemeral keys
    - Elliptic Curve Cryptography
    - EdDSA (Edwards Digital Signature Algorithm)
- Hashing
- Password-Based Key Derivation

3.3.2. Challenges

The main challenge with designing this library was getting around Libgcrypt's use of s-expressions and multi-precision integers (mpis). S-expressions are treelike structures used to store large pieces of data. Libgcrypt utilizes these data structures because some of the keys and information it needs to do cryptographic operations are so large the C programming language cannot store them as strings. However, they are not used for anything else in the modern workplace and are very cumbersome to work with. So, I decided to let the user keep all their keys/authentication information as strings and to handle the s-expression conversion myself before and after encryption.

This was very hard to accomplish, as there is little online about working with s-expressions in cryptography and next to no information in the Libgcrypt manual. Multi-precision integers (MPIs) are typically present within s-expressions and are also used to represent enormous numbers. These were challenging to work with because they also lack online resources, and it is necessary to convert parts of your string to an MPI before converting them to an s-expression.

Another challenge with the design of this library had to do with elliptic curves. Elliptic curves are very useful in cryptography for encryption, key generation and Diffie-Hellman exchanges. However, Libgcrypt does not provide methods to accomplish these protocols directly. It does, however, allow one to do curve multiplication over two different curves. This meant doing the key generation or encryption by hand in the library. This was challenging because these methods took MPIs or strings as arguments, which means converting a key to a string or extracting a specific MPI from an s-expression in order to do this math. This was a tedious task and took a lot of testing and trial and error

One main challenge with this library design was the lack of resources online about dealing with Libgcrypt. The online community for libgcrypt is extremely small and mostly contains basic questions about its operations, rather than the advanced methods the library dealt with.

3.3.3. Solutions

To solve the challenge of converting s-expressions to and from strings, I had to figure out an advanced use of Libgcrypt's methods for s-expression conversion. This method takes in certain key words as arguments and those key words don't have much explanation in the documents. Through trial and error, I had to utilize different combinations of these key words to find out what they mean and determine the right arguments for my purposes. There were no resources regarding this online, but I was able to figure it out through essentially brute force. Another solution to the issue of converting s-expression to strings was to add an intermediary step of converting to a multi precision integer before converting to or from string. This means

converting an s-expression to mpi, then to string or vice versa. The libgrypt library doesn't say anything about this, but it is a crucial step I discovered for proper conversions.

4. RESULTS

The result was a robust cryptographic library that had no memory leaks and accomplished all basic cryptographic functions plus a few advanced ones. This library successfully allowed the user to generate and store keys, authentication data, hashed passwords and other items as user-friendly strings. The library was also extremely easy to use, cutting the amount of user code needed to accomplish a basic crypto scheme by about two thirds. The library was also successfully interchangeable with popular libraries like Libgcrypt and Libsodium, which allowed the user to encrypt data, send it over a tcp socket to another file that would decrypt it with a mainstream library, and vice versa.

5. CONCLUSION

The result of this project is a robust cryptographic library that has all the necessary features one could need to accomplish just about any cryptographic protocol they may want to implement. This library's development was important because not many cryptographic libraries are intuitable and require a steep learning curve. Not to mention, many cryptographic libraries that claim to be easy to use are not very powerful and provide little to no customization to the user. The library built successfully addressed the gap between usability and power. For the client, this means less lines of code, less time spent debugging code, and overall increased productivity.

6. FUTURE WORK

Next possible steps for this project would be to include support for more elliptic curves, this would include the actual detailed low-level implementation of such curves. Additionally, curve conversion could be implemented to allow users to accomplish authenticated encryption and signature.

Another area of expansion could add support for additional data types. Right now, users can give the library strings to go through a cryptographic protocol (encryption, signatures, hashing, etc.). It could be convenient for the user to be able to send in integers, doubles, or other popular types to the library.

REFERENCES

1. Tygar, J., & Whitten, A. (1999). (rep.). *Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0* (pp. 1–15).
2. The LIBGCRYPT Reference Manual. (n.d.) Top (The Libgcrypt Reference Manual). https://www.gnupg.org/documentation/manuals/gcrypt/