Tropyc: A Pattern Language for Cryptographic Software*

Alexandre M. Braga Cecília M. F. Rubira Ricardo Dahab

State University of Campinas
Institute of Computing
P.O. Box 6176
13081-970 Campinas-SP-Brazil
phone/fax:+55+19+7885842
e-mail:{972314,cmrubira,rdahab}@dcc.unicamp.br

Abstract

This work describes *Tropyc*, a pattern language for cryptographic software. Nine patterns are described: *Information Secrecy*, *Message Authentication*, *Message Integrity*, *Sender Authentication*, *Secrecy with Authentication*, *Secrecy with Signature*, *Secrecy with Integrity*, *Signature with Appendix*, and *Secrecy with Signature with Appendix*. These patterns are classified according to four fundamental objectives of cryptography and compose a closed set of patterns for that domain. These patterns have the same dynamic behavior and structure. We abstracted these aspects in a generic object-oriented *Cryptographic Metapattern*.

Key words: cryptography, pattern language, design patterns, software architecture, object orientation.

1 Introduction

Traditionally, cryptography has been concerned with algorithms. Nowadays, cryptographic techniques are been widely used in many applications, such as word processors, spreadsheets, databases, and electronic commerce systems. The widespread use of cryptographic techniques and the present interest and research on software architectures and patterns led to cryptographic software architectures and cryptographic patterns.

In this work we present a set of nine cryptographic patterns classified according to the objectives of cryptography, which are described in Appendix A, and organized as a pattern language for cryptographic software. The patterns are presented here in a simple way and our focus is on the pattern language. This paper is targeted to software engineers who have to deal with cryptography based security requirements of applications, but do not have much experience in cryptography. People looking for general software architectures to their cryptographic software or components are another group of interest. This work is organized as follows. Section 2 proposes *Tropyc*, a

*Copyright 1998, Alexandre M. Braga. Permission is granted to copy for the PLoP-98 conference.

pattern language for cryptographic software. The completeness of the set of cryptographic patterns and their relations with other patterns are investigated in Section 3. Conclusions and future work are in Section 4. In order to define some terms, a brief introduction to cryptography is shown in Appendix A. Also, we provide structural and dynamic models for all patterns in Appendix B.

2 A Pattern Language for Cryptographic Software

In this Section we propose *Tropyc*, a pattern language for cryptographic software. Our proposal focuses on two goals: (*i*) the description of cryptographic services as patterns and (*ii*) the organization of these cryptographic patterns as nodes of a directed acyclic graph, in which a walk documents a sequence of design decisions. Table 2.1 summarizes all patterns, their scopes according to cryptographic objectives, and their purposes.

We have to deal with two kinds of forces when instantiating a cryptographic pattern: (*i*) the forces produced by the trade-offs one must consider when deciding whether to use a specific cryptographic service or not; and (*ii*) the forces one must consider when deciding to structure its cryptographic software or component as an instantiation of our *Cryptographic Metapattern*. Thus, we decide to present these distinct sets of forces separately, in order to highlight this distinction of concerns: the Section "Forces" of each cryptographic pattern, except the *Cryptographic Metapattern*, focuses on the trade-offs of whether or not a cryptographic service or a combination of services should be used. The *Cryptographic Metapattern* Section "Forces" focuses on the second type, that is, those forces acting over the instantiation of our generic structure.

Another important concern is how well *Tropyc* is supported by the cryptographic infrastructure services (see Appendix A for information about cryptographic infrastructure services). The variety of algorithms, protocols and hardwares available to these auxiliary services can result in a new set of trade-offs and forces. We call these forces auxiliary because they do not act directly over the choice of a cryptographic pattern, but can strongly influence its implementation, since they are related to features available on a specific cryptographic library. In order to highlight these auxiliary forces, some of the cryptographic patterns present them into a Section "Implementation Factors". Traditionally, the two ends of a communication channel are called Alice and Bob, Eve is usually an adversary eavesdropping the channel.

2.1 Pattern Language Summary

Tropyc supports the decision making process of choosing which cryptographic services address applications requirements and user needs, offering a set of ten closely related cryptographic design patterns. When on-line communication or exchange of information through files is used, sometimes, due to great sensitiveness of data, it is necessary to guarantee its (*Information*) *Secrecy*. However, secrecy alone does not prevent either data modification or replacement. Particularly in on-line communication, granting *Message Integrity* and (*Message*) *Authentication* is also important. There are some contexts in which it is necessary to prevent an entity from denying her actions or commitments. For example, some form of *Sender Authentication* is necessary when purchasing electronic goods over the Internet.

Sometimes, the cryptographic based security requirements of applications lead developers to compositions of the basic services in order to provide *Secrecy with Integrity*, *Secrecy with Authentication*, *Secrecy with Signature*. Cryptography is time consuming, so algorithm performance is always important. *Sender Authentication* can be speeded up if a *Signature with Appendix* is used. For efficiency, it is recommended that *Secrecy with Signature with Appendix* be implemented instead of *Secrecy with Signature*. All of the above situations, especially those with combined services, need a flexible and reusable software architecture. This architecture is common to all cryptographic patterns and could be abstracted as a generic *Cryptographic Metapattern*.

Figure 1 is a directed acyclic graph of dependences among patterns. An edge from pattern A to pattern B shows that pattern A generates pattern B. A pattern that is pointed at by more than one edge has as many generators as the number of edges arriving in it. The *Cryptographic Metapattern* generates the microarchitecture for the four basic patterns. All other patterns are generated from combinations of these. Thus, all nine cryptographic patterns instantiate the *Cryptographic Metapattern*. In the case of the *Secrecy with Signature with Appendix* pattern, we decided to derive it from two combined patterns, instead of using three basic patterns, because we feel a two generator combination is more intuitive than a three generator combination. A walk on the graph is directed by two questions: What cryptographic services should I use to address application requirements and user needs? And how should I structure the cryptographic component to obtain easy reuse and flexibility?

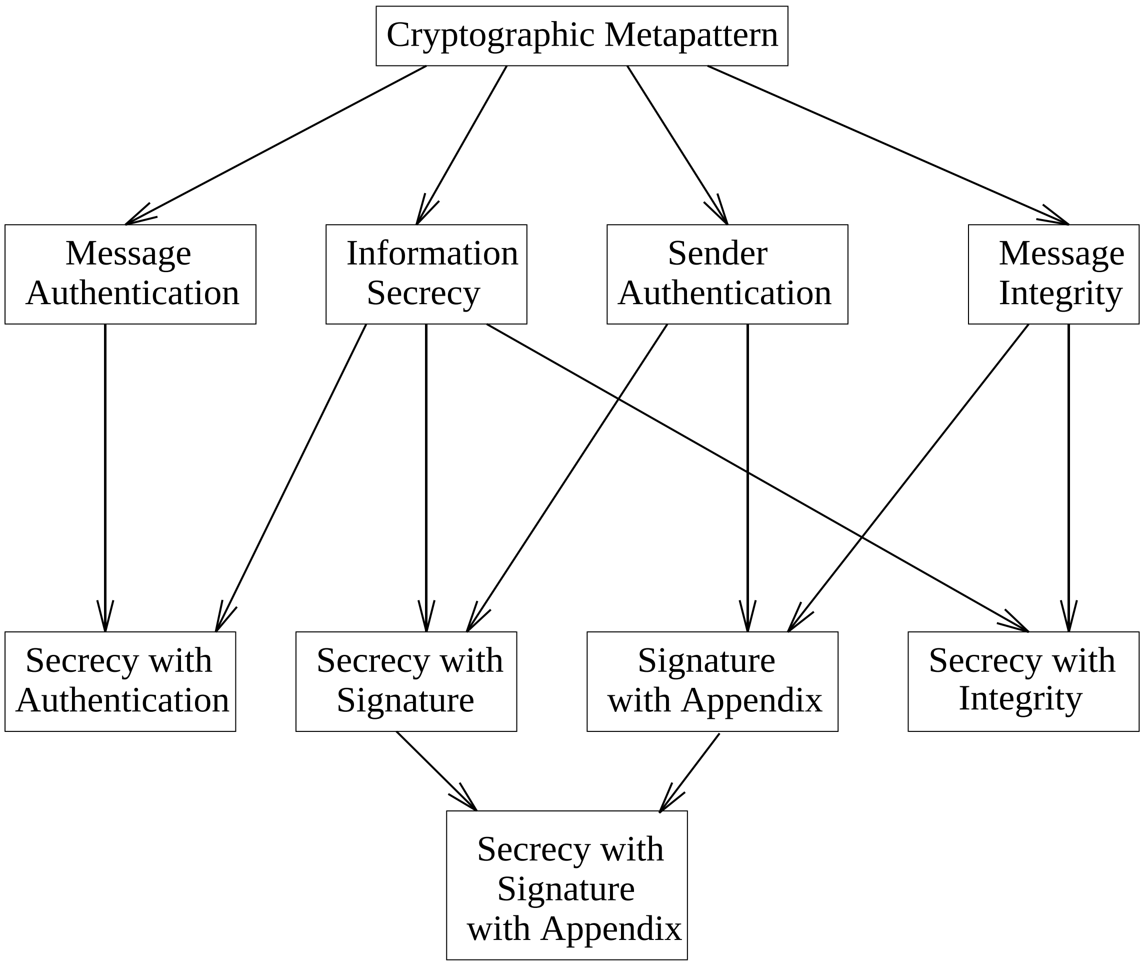

Figure 1: Cryptographic Design Patterns and Their Relationships.

Pattern	Scope	Purpose
Information Secrecy	Secrecy	keep the secrecy of information
Message Integrity	Integrity	avoid corruption of a message
Message Authentication	Authentication	authenticate the origin of a message
Sender Authentication	Non-repudiation	avoid refusal of a message
Secrecy with Authentication	Secrecy	prove the authenticity of a secret
Secrecy with Signature	Secrecy	prove the authorship of a secret
Secrecy with Integrity	Secrecy	keep the integrity of a secret
Signature with Appendix	Non-repudiation	separate message from signature
Secrecy with Signature with Appendix	Secrecy	separate secret from signature
Cryptographic Metapattern	Generic	Define a generic software architecture to cryptography

Table 2.1: The Cryptographic Design Patterns and Their Purposes.

2.2 Information Secrecy

Context Alice wants to send messages to Bob. She wants to keep these messages secret from Eve, whom Alice suspects may be trying to read her messages.

Problem How can Alice send a message to Bob in such a way that Eve cannot possibly read its content?

Forces

- Encryption is slow. If performance is important, its use must be evaluated carefully. For example, the cost of encrypting/decrypting messages must not be greater than the intrinsic value of the message.
- The security of an encrypted information is in the encryption key. Thus, the cost of breaking the cryptographic keys must be much greater than the intrinsic value of the message.
- The strength of a cryptosystem is based on the secrecy of a long key. However, the longer the key, the slower is the algorithm.

Solution This pattern supports encryption and decryption of data. Alice and Bob previously agree on a cryptographic engine and on a shared secret key (if public key cryptography is used, Bob must first obtain Alice's public key). Bob enciphers the message and sends it to Alice. Alice deciphers the enciphered message and recovers the original message. The structural and dynamic models for that pattern are shown in Section B.1.

Implementation Factors

- Private or secret keys must be kept protected from unauthorized copy or modification.
- An infrastructure to distribute or make public keys broadly available is necessary.

Example A common use of this pattern is in the encryption of electronic mail messages [Her97].

2.3 Message Integrity

Context Alice sends large messages to Bob. He wants to verify the integrity of received messages that Eve may have modified or replaced. Alice and Bob do not share cryptographic keys, so they cannot sign messages or authenticate them with Message Authentication Codes (MACs) (see Section A.1).

Problem How can Bob determine if a message was modified or replaced after being sent and before its arrival to him?

Forces

- Integrity of messages can be obtained by sending each message at least twice and comparing the copies. Furthermore, sending short messages twice may be less expensive than computing and sending a relatively large Modification Detection Code (MDC) (see Section A.1).
- It is necessary to verify a relatively small MDC to determine whether a large amount of data has been modified or not.
- MDCs by themselves do not guarantee the authorship of a message.

Solution Alice and Bob agree to use a MDC. Alice computes the MDC of the message and sends both message and MDC to Bob. Bob computes the MDC of the message and compares it with that received from Alice. If they match, the message is genuine. The structural and dynamic models for that pattern are shown in Section B.2.

Implementation Factors

- MDCs which are not themselves authentic do not provide any guarantees whether a message has been replaced or not.

Example Two common uses of MDCs are detection of file modification caused by viruses and generation of passphrases to produce cryptographic keys. Also, MDCs could be used as unique identifiers of electronic coins in electronic commerce applications based on hash chains [AJSW97].

2.4 Message Authentication

Context Alice and Bob want to exchange messages, but they cannot distinguish their own messages from the ones Eve may have included into the communication channel. Also, they have the ability to share secrets in a secure way.

Problem How can genuine messages be distinguished from spurious ones?

Forces

- A message can only be authenticated if an information recognized only by the communicating parts is intrinsically associated with it. If the authentication information is a function of both data and a secret, the replacement of messages or parts of them can be detected.
- Authorship cannot be granted since both sides can compute valid MACs.
- The value of the data must be greater than the cost of granting its authenticity.

Solution Alice and Bob agree previously on a shared secret key and a cryptographic algorithm to generate Message Authentication Codes (MACs). Alice computes the MAC of the message and sends both, message and MAC, to Bob. Bob computes the MAC again and compares it with the one received from Alice. If they match, the message is genuine and must be sent by Alice, because, other than Bob, only Alice knows the secret key and can compute the correct MAC. A bonus when using this pattern is the implicit integrity of messages. The structural and dynamic models for that pattern are shown in Section B.3.

Implementation Factors

- A secure means to exchange and keep a secret is necessary.
- The security of MACs is in the secrecy of the key and in the one-wayness of the cryptographic hash function.

Example MACs could be used in the authentication of IP packages over the Internet [CGHK98].

2.5 Sender Authentication

Context Alice sends messages to Bob, but they cannot distinguish their own messages from the ones Eve may include into the communication channel. Furthermore, Alice can later dispute the authorship of messages actually sent by her. In such a situation, Bob cannot prove to a third party that only Alice could have send that message. Also, Alice has a public/private key pair, or the ability to create key pairs, and her public key is widely available.

Problem How to guarantee that messages have a genuine and authentic sender in such a way that Alice cannot repudiate a message Bob believes was sent by her?

Forces

- Fake digital messages can be easily generated or copied. Thus, explicit measures must be taken in order to associate a message with its legitimate sender. Digital signatures (see Section A.1) provide the means with which one can tie the identity of an entity with a message produced by that entity in such a way that the legitimate sender of a signed message cannot deny its authorship.
- Signatures are usually as large as the data being signed.
- The cost of granting the authorship of a message cannot be greater than its intrinsic value.

Solution Alice and Bob agree on the use of a public key digital signature protocol and Bob has Alice's public key. Alice enciphers a message with her private key to sign it and sends the signed message to Bob. He deciphers the signed message with Alice's public key in order to verify it. If the enciphered message makes sense to Bob, then, since only Alice's private key could have been used to generate a meaningful message after decipherment by Bob, it must be true that Alice is the sender of that message. The structural and dynamic models for that pattern are shown in Section B.4.

Implementation Factors

- Public key cryptographic algorithms are generally used to generate digital signatures.
- A secure means of storing the author's private key is necessary.
- An infrastructure to distribute or make public keys broadly available is necessary.
- Verifying the authorship of a message is based solely on the secrecy of the author's private key.

Example Digital signatures are used in electronic commerce applications in the authentication of customers and merchants [AJSW97, HY97]. Also, they could be used to guarantee the authenticity of information obtained over the Internet [HN98].

2.6 Secrecy with Authentication

Context Alice and Bob use public key cryptography to exchange encrypted messages. Eve may intercept messages, but she cannot read their contents. However, she can replace or modify these messages in such a way that Alice and Bob cannot detect these modifications or replacements.

Problem How can Alice authenticate an encrypted message without loss of secrecy?

Forces

- In public key cryptography, valid encrypted messages can potentially be generated by anyone who has access to the public key.
- If Alice and Bob use secret key cryptography for encryption and they are the only ones who share that secret, *Message Authentication* is redundant and useless, except for granting integrity implicitly; but this can be obtained using *Secrecy with Integrity.*
- *Message Authentication* and encryption are independent services and providing one does not necessarily imply the other.
- *Message Authentication* restricts the number of entities who can produce genuine encrypted messages, but do not grant authorship, since usually more than one entity knows the secret key.
- *Message Authentication* inserts a new step in both the encryption and decryption processes in order to compute and verify a MAC, which can result in a decrease of performance.

- The combination of secrecy and authentication can be easily done when using a design in which the modularity of components facilitates reuse.

Solution Two previous cryptographic patterns are combined to solve this problem: *Information Secrecy* and *Message Authentication*. The MAC must be computed over the original unencrypted message. Both encrypted message and MAC are sent to Bob. The secret key used to compute the MAC must be different from that used to encipher/decipher. The structural and dynamic models for that pattern are shown in Section B.5.

Example Secrecy and authentication could be combined in order to secure IP packages over the Internet [CGHK98].

2.7 Secrecy with Signature

Context Alice and Bob exchange encrypted messages, but they cannot grant the authorship of an encrypted message. Furthermore, Eve can modify, replace or include messages into the communication channel in such a way that Alice and Bob cannot detect the spurious messages. Alice and Bob already share keys for secrecy purposes.

Problem How can Bob prove the authorship of an encrypted message without loss of secrecy in such a way that its integrity and origin authentication is also implicitly granted?

Forces

- Anyone who can produce valid encrypted messages can potentially deny their authorship in the future.
- Proof of authorship and secrecy are independent services, the existence of one not implying the other's.
- *Sender Authentication* can grant authorship of encrypted messages by inserting an intermediate step into the encryption/decryption processes, causing a loss of efficiency.
- The combination of secrecy and digital signatures can be easily done when using a design in which the modularity of components facilitates reuse.

Solution Two previous cryptographic patterns are combined to solve this problem: *Information Secrecy* and *Sender Authentication*. Alice signs a message with her private key, encrypts the signed message with Bob's public key (alternatively, the signed message could be encrypted with a shared secret key) and sends it to Bob. Bob deciphers the encrypted message with his private key (or a shared secret key) and verifies the signed message with Alice's public key. The structural and dynamic models for that pattern are shown in Section B.6.

Implementation Factors

- It is recommended that two pairs of cryptographic keys are used with this pattern. One pair used to encryption purposes and the other used to signing and verification.

Example When sending his/her credit card number over the Internet, a user requires its secrecy. On the other hand, a vendor upon receiving such a number, needs the assurance of non-repudiation by the sender in the future.

2.8 Secrecy with Integrity

Context Alice and Bob exchange encrypted messages, but they cannot detect modifications or replacements of encrypted messages possibly done by Eve. Also, they do not want to share a secret key for authentication purposes alone.

Problem How to preserve the integrity of an encrypted message without loss of secrecy?

Forces

- Errors during transmission and malicious modifications or replacements can cause valid data to become completely garbled, or change its meaning, after decryption.
- Keep data integrity and secrecy are independent services, the existence of one not implying the other's.
- The computation and verification of MDCs can grant the integrity of encrypted data, but can also cause a decrease of performance.
- Authentication and proof of authorship of encrypted messages are not necessarily implied by the use of secrecy and integrity.
- The combination of secrecy and integrity can be easily done when using a design in which the modularity of components facilitates reuse.

Solution Two previous patterns are combined to solve this problem: *Information Secrecy* and *Message Integrity*. The MDC must be computed over the original unencrypted message. Both encrypted message and MDC are sent to Bob. This pattern only needs one public/private key pair for encryption purposes. The structural and dynamic models for that pattern are shown in Section B.7.

Example Secure file transfer requires file integrity and secrecy.

2.9 Signature with Appendix

Context Alice and Bob sign messages they exchange in order to prevent modifications or replacement and to provide *Sender Authentication*. They need to manage limited storage and processing resources. However, the messages they exchange are very large and also produce large signatures.

Problem How to reduce the storage space required for a message and its signature while increasing the performance of the digital signature protocol?

Forces

- When no technique to reduce signature size is used, digital signatures are usually as large as the data being signed. However, if messages are small, the inclusion of a new computation step to reduce the signature size is not necessary. An important trade-off is the relation between the impact of additional computations over performance and the ratio between the size of messages and their signatures.
- The combination of MDC computations with *Sender Authentication* does not increase the security of digital signature algorithms.
- The combination of *Sender Authentication* and integrity can be easily done when using a design in which the modularity of components facilitates reuse.

Solution Two patterns are combined to solve this problem: *Sender Authentication* and *Message Integrity*. This pattern implements a digital signature protocol over a message hash value, which is a MDC. Alice computes a hash value of the message and signs it. Both message and signed hash value are sent to Bob. Bob decrypts the signature and recovers the hash value. He then computes a new hash value and compares it with the one recovered from the signature. If they match, the signature is true. The structural and dynamic models for that pattern are shown in Section B.8.

Example When a user of an Internet application must digitally sign information, for efficiency reasons, it is better to produce small signatures [CGHK98].

2.10 Secrecy with Signature with Appendix

Context Alice and Bob exchange encrypted signed messages to prevent modifications or replacement and to achieve secrecy and sender authentication. They need to manage limited storage and processing resources. However, the messages they exchange are very large and produce large signatures. Also, they already have a time delay due to the encryption process.

Problem How to reduce the memory necessary to store a message and its signature, while increasing system performance, without loss of secrecy?

Forces

- The inclusion of a new processing step to reduce the signature size within a computation that already has two processing phases, one to encrypt/decrypt data and another to compute or verify a signature, is a difficult decision. Again small losses of performance must be negligible in order to save a relatively large amount of space and reduce the amount of data to be transmitted.
- The combination of *Information Secrecy* and *Signature with Appendix* can be easily done when using a design in which the modularity of components facilitates reuse.

Solution Two patterns are combined to solve this problem: *Secrecy of Information* and *Signature with Appendix*. Alice computes a hash value of the message and signs it with her private key. She then encrypts the original message with Bob's public key. Both encrypted message and signed hash value are sent to Bob. He deciphers the encrypted message with his private key and decrypts the signed hash value with Alice's public key. Bob computes a new hash value of the message and compares it with that received from Alice. If they match, the signature is true. It is important to remember that this technique is also valid if secret key cryptography is used. The structural and dynamic models for that pattern are shown in Section B.9.

Example Electronic forms usually contain a lot of sensitive customer information which requires secrecy and non-repudiation. This pattern could be used to increase the performance of secure Internet applications that use large electronic forms.

2.11 Cryptographic Metapattern

Context Two objects, Alice and Bob, exchange data through messages. They need to perform some kind of cryptographic transformations, alone or in combinations, over the data. They want a cryptography component that is flexible and that could be easily reused with other cryptographic transformations.

Problem How to design a flexible object-oriented microarchitecture for a cryptographic design pattern in order to increase component reuse?

Forces

- Traditionally cryptography has been concerned with algorithms. However, all cryptographic transformations have a common behavior that could be generalized in a flexible design.
- Cryptography is time consuming and *Ad hoc* implementations of cryptographic facilities could have a better performance than well designed ones, caracterized by insertion of indirections into the code, which slows down the execution.
- Well structured software architectures facilitate reuse and are flexible to future adaptations. Flexible and adaptable systems with cryptographic based security requirements can be more easily obtained when cryptographic algorithms are decoupled from their implementations, and these two are, in turn, decoupled from the cryptographic services those systems use.

Solution Alice performs a cryptographic transformation on the data before sending it to Bob. Bob receives the message and performs the inverse transformation to recover the data. Alice and Bob must agree about what transformation to perform and share or distribute keys, if necessary. The class diagram shown in Figure 2 generalizes the cryptographic transformation with an abstract transformation interface and distinguish the Sender and Receiver roles from the Codifier and Decodifier roles.

The *Cryptographic Metapattern* is a higher level abstraction for all cryptographic patterns. All cryptographic patterns instantiate the metapattern structure and dynamics. The metapattern has two template classes, Alice and Bob, and two hook classes, Codifier and Decodifier, as shown in Figure 2. The class Codifier has a hook method $f(x)$, which performs a cryptographic transformation over x. The class Decodifier has a hook method $g(x)$, which performs the reverse transformation, $x = g(f(x))$. The transformation and its reverse are based on the same cryptographic algorithm. Figure 3 shows the metapattern dynamic behavior. All patterns presented here assume the existence of on-line communication. However, they can be easily adapted to deal with file storage and recovery. In such a situation, the send and receive messages could be replaced by store and recovery ones, respectively. Also, the reference Alice has to Bob can be avoided.

Example The dynamic and structural models of all previous patterns are instantiations of the *Cryptographic Metapattern*, as shown in Appendix B.

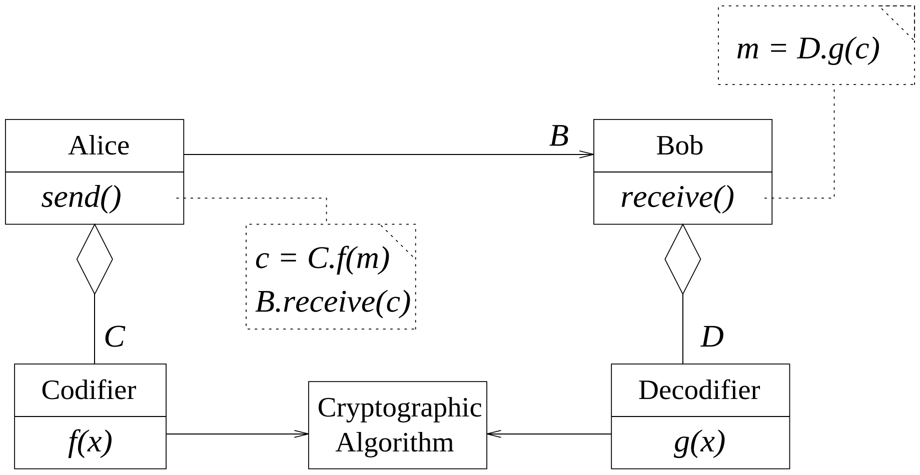

Figure 2: Cryptographic Metapattern Structure.

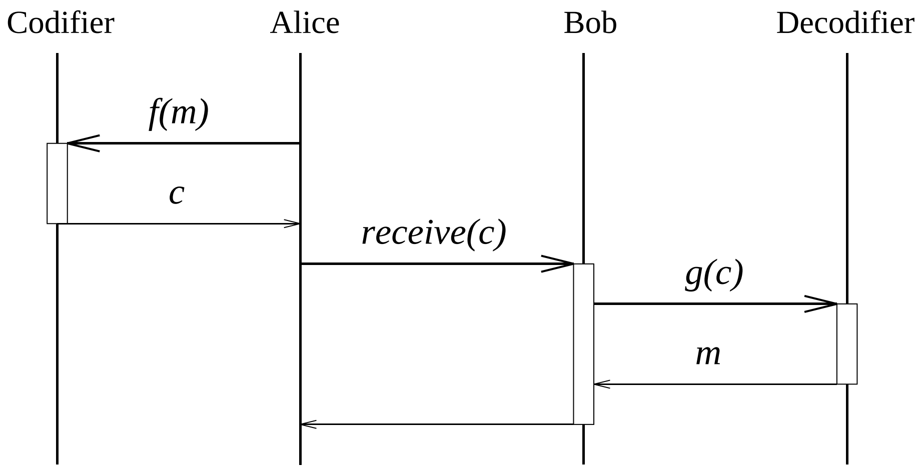

Figure 3: Cryptographic Metapattern Dynamics.

3 Completeness of the Set and Relationships with Other Patterns

We consider our pattern language to be complete and closed into the cryptographic domain for two reasons: (*i*) The nine patterns, except the *cryptographic Metapattern*, represent all possible valid combinations of the four cryptographic objectives. Thus, other patterns that might emerge will be support patterns and will not offer new services based on the cryptographic objectives. However, variations of the cryptographic patterns other than the ones proposed here are possible. Furthermore, auxiliary services such as key generation, key agreement and key distribution, and auxiliary techniques such as digital certification, are not considered as cryptographic patterns, because they do not provide general solutions as software microarchitectures to the problems addressed by the set of cryptographic objectives. Figure 4 shows, as a symmetric matrix, all possible valid combinations of cryptographic services and the alternative ways of producing some cryptographic patterns. Some combinations do not add any new features to their already generated patterns so they are not significant; that is, each valid combination does not support repeated cryptographic objectives; (*ii*) Many generic cryptographic service APIs support this set of cryptographic patterns to some extent, and all cryptographic patterns have at least one instantiation in at least one cryptographic service API. The cryptographic patterns are widely used in many applications and supported by many cryptographic APIs, at least implicitly. We reference just a few [Deg97, Kal95, CSS97, HY97, AJSW97, Her97, CGHK98, HN98].

Some well known design patterns can be used when designing a cryptographic pattern. The *Strategy* [GHJV94, 315] pattern can be used to obtain algorithm independence. The *Bridge* [GHJV94, 151] pattern can be used to obtain implementation independence. The *Abstract Factory* [GHJV94, 87] pattern can be used in the previous negotiation step to decide which algorithm or implementation to use. The *Observer* [GHJV94, 293], *Proxy* [GHJV94, 207], and *Client-Dispatcher-Server* [BMR$^+$96, 323] patterns can be used to obtain location transparency. The *Forwarder-Receiver* [BMR$^+$96, 307] pattern could be combined with the cryptographic patterns in order to offer secure and transparent interprocess communication, in such a way that Alice becomes part of the Forwarder and Bob is incorporated into the Receiver. The *State* [GHJV94, 305] pattern could also be used to provide state dependent behavior, such as turning the security of the channel on and off. Patterns related to security aspects of applications are proposed by Yoder and Barcalow [YB97]. These patterns are concerned with higher level application security aspects. In fact, lower level security, such as cryptography, is not treated and a pattern to encapsulate these lower level security aspects in a *Security Access Layer* [YB97, 16] is proposed.

4 Conclusions and Future Work

Cryptographic support is becoming a default feature in many applications. In order to facilitate the design, implementation and reuse of flexible and adaptable cryptographic software, one should consider the architectural aspects of cryptographic components and the patterns that emerge from them. In this work, we present a pattern language for cryptographic software. This pattern language can be used to guide the decision making process for the design of cryptographic features. It can also be used in the design and documentation of a cryptographic object-oriented framework.

5 Acknowledgments

This work is partially supported by CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico*) and FAPESP (*Fundação de Amparo à Pesquisa do Estado de São Paulo*), grant 97/11128-3 for Alexandre Melo Braga and 96/15329 for LSD-IC-UNICAMP (*Laboratório de Sistemas Distribuídos, Instituto de Computação , Universidade Estadual de Campinas*). We would like to thank Robert Orenstein, whose shepherding was very important during the writing of the final version of this paper.

	M.A.	I.S.	S.A.	M.I.	S.w.A.	S.w.I.	S.w.S.	S.w.Ap.	S.S.w.A.
M.A.	■	S.w.A.	⬆	⬅	■	S.w.A.	⬆	⬆	■
I.S.	S.w.A.	■	S.w.S.	S.w.I.	■	■	■	S.S.w.A.	■
S.A.	⬅	S.w.S.	■	S.w.Ap.	S.w.S.	S.S.w.A.	■	■	■
M.I.	⬆	S.w.I.	S.w.Ap.	■	⬆	■	⬆	⬆	■
S.w.A.	■	■	S.w.S.	⬅	■	⬅	⬆	S.S.w.A.	■
S.w.I.	S.w.A.	■	S.S.w.A.	■	⬆	■	S.S.w.A.	S.S.w.A.	■
S.w.S.	⬅	■	■	⬅	■	S.S.w.A.	■	S.S.w.A.	■
S.w.Ap.	⬅	S.S.w.A.	■	⬅	S.S.w.A.	S.S.w.A.	S.S.w.A.	■	■
S.S.w.A.	■	■	■	■	■	■	■	■	■

M.A. - Message Authentication
I.S. - Information Secrecy
S.A. - Sender Authentication
M.I. - Message Integrity
S.w.A. - Secrecy with Authetication
S.w.I. - Secrecy with Integrity
S.w.S. - Secrecy with Signature
S.w.Ap. - Signature with Appendix
S.S.w.A. - Secrecy with Signature with Appendix

■	Not significant
⬅	Supported by line
⬆	Supported by Column

Figure 4: All Possible Valid Cryptographic Patterns.

References

[AJSW97] N. Asokan, Philippe A. Janson, Michael Steiner, and Michael Waidner. The State of the Art in Electronic Payment Systems. *IEEE Computer*, pages 28–35, September 1997.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons Ltd., Chichester, UK, 1996.

[CGHK98] Pau-Chen Cheng, Juan A. Garay, Amir Herzberg, and Hugo Krawczyk. A Security Architecture for the Internet Protocol. *IBM Systems Journal*, 37(1):42–60, 1998.

[CSS97] Common Security Services Manager Application Programming Interface, Draft 2.0. http://www.opengroup.org/public/tech/security/pki/index.htm, June 1997.

[Deg97] Mary Degeforde. Java Cryptography Architecture API Specification and Reference. http://java.sun.com/products/JDK1.1/docs/guide/security/CryptoSpec.html, February 1997.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Publishing Company, April 1994.

[Her97] Michael Herfert. Security-Enhanced Mailing Lists. *IEEE Network*, pages 30–33, 1997.

[HN98] Amir Herzberg and Dalit Naor. Surf'N'Sign: Client Signatures on Web Documents. *IBM Systems Journal*, 37(1):61–71, 1998.

[HY97] Amir Herzberg and Hilik Yochai. Minipay: Charging per Click on the Web. *Computer Networks and ISDN Systems*, 1997.

[Kal95] B. Kaliski. Cryptoki: A Cryptographic Token Interface, Version 1.0. http://www.rsa.com/rsalabs/pubs/PKCS/html/pkcs-11.html, April 1995.

[MvOV96] Alfred J. Menezes, Paul C. van Orschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Sch96] Bruce Schneier. *Applied Cryptography — Protocols, Algorithms , and Source Code in C*. John Wiley and Sons, 2nd edition, 1996.

[YB97] Joseph Yoder and Jeffrey Barcalow. Application Security. *PLoP'97 Conference, Washington University Technical Report 97-34*, 1997.

A Basic Cryptographic Concepts

Historically associated to encipherment, modern cryptography is a broader subject, encompassing the study and use of mathematical techniques to address information security problems, such as confidentiality, data integrity and non-repudiation. Usually [MvOV96], four objectives of cryptography are considered: confidentiality, integrity, authentication, and non-repudiation. Accordingly, there are four basic cryptographic services: (*i*) encipherment/decipherment to obtain secrecy or privacy, (*ii*) MDC (Modification Detection Code) generation/verification, (*iii*) MAC (Message Authentication Code) generation/verification, and (*iv*) digital signing/verification. These four services can be combined in specific and limited ways to produce more specialized services. Some possible combinations of services are secrecy with authentication and secrecy with digital signatures.

The four cryptographic objectives can be stated as:

- Confidentiality is the ability to keep information secret except from authorized users.
- Data integrity is used to guarantee that the information has not been modified without permission, which includes the ability to detect unauthorized manipulation.
- Data authentication can be obtained in two ways:
  - Entity authentication, which is used when two parts need to achieve mutual assurance about each other's identity before initiating a communication.
  - Information authentication, which corresponds to authentication of the origin of an information transmitted through an insecure communication channel.
- Non-repudiation is the ability to prevent an entity from denying its actions or commitments in the future.

A.1 Cryptographic Services

Secret or symmetric key cryptography is the set of cryptographic techniques in which a single key is used both to encrypt and decrypt data. The key is a shared secret between two entities. In public key cryptography, a pair of different keys is used, one key for encryption, the other for decryption. The encryption key is publicly known, so it is called the public key. The corresponding decryption key is a secret known only to the key pair owner, so it is called the private key. One fundamental feature of public key cryptography is that it is computationally infeasible to deduce the private key from the knowledge of the public key.

Figure 5 shows a typical cryptosystem. Traditionally, the two ends of a communication channel are called Alice and Bob. Eve is an adversary eavesdropping the channel. Alice wants to send an encrypted message to Bob; she encrypts message m, the plain text, with an encryption key $k1$ and sends the encrypted message c, the cipher text, to Bob. Bob receives the encrypted message and deciphers it with a decryption key $k2$ to recover m. If public-key cryptography is used, Alice uses Bob's public key to encrypt messages to Bob, and Bob uses his private key to decrypt messages sent from anyone who used his public key. However, if symmetric-key cryptography is used, Alice and Bob share a secret key used to encrypt and decrypt messages they send to each other. That is, $k1 = k2$. Two good sources of information about cryptography are [MvOV96, Sch96].

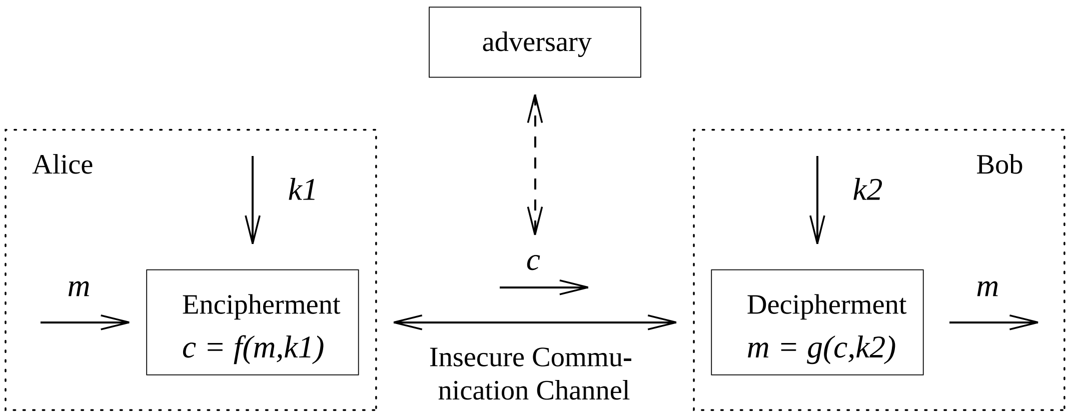

Figure 5: A Typical Cryptosystem.

A hash function is a mathematical function that takes as input a stream of variable length and returns as a result a stream of fixed length, usually much shorter than the input. One-way hash functions are hash functions in which it is computationally easy to compute the hash value of an input stream, but it is computationally difficult to determine any input stream corresponding to a known hash value. A cryptographic hash function is a one-way collision-resistant hash function; that is, it is computationally difficult to find two input streams that result in the same hash value. Hash values produced by cryptographic hash functions are also called Modification Detection Codes (MDCs for short) and are used to guarantee data integrity. Message Authentication Codes (MACs for short) are usually implemented as hash values generated by cryptographic hash functions which take as input a secret key as well as the usual input stream. MACs are used to provide data authentication and integrity implicitly.

Digital Signatures are electronic analogs of traditional handwritten signatures of a document, which serve as the signer's agreement to the information it contains, and also as evidence that could be shown to a third party in case of repudiation. A basic protocol of digital signatures based on public key cryptography is:

1. Alice encrypts a message with her private key to sign it.
2. Alice sends the signed message to Bob.
3. Bob decrypts the received message with Alice's public key to verify the signature.

Digital signatures must provide the following features:

- They are authentic. When Bob verifies a message with Alice's public key, he knows she signed it.
- They are unforgeable. Only Alice knows her private key.
- They are not reusable. The signature is a function of the data being signed, so it cannot be used with other data.
- They cannot be repudiated. Bob does not need Alice help in order to prove she signed a message.
- The signed data is unalterable. Any modification of the data invalidates the signature verification.

A.2 Common Attacks

Eve can attack a cryptosystem in four ways:

1. She can eavesdrop the channel. Eavesdropping an open channel is easy. However, in order to understand eavesdropped messages of a cryptographically secured channel, the key (or keys) being used by Alice and Bob are required.

2. She can re-send old messages. This attack is possible if messages do not have temporal uniqueness, which can be obtained using timestamps or by changing keys periodically.

3. She can impersonate one of the communicating ends of the channel. In such a case, Eve plays the role of Alice or Bob, either by: (*i*) deducing a secret key or (*ii*) by successfully substituting her public key for Alice's (Bob's) without Alice's (Bob's) knowledge.

4. She can play the role of the man-in-the-middle. In order to perform the man-in-the-middle attack successfully, Eve must: (*i*) have obtained the private keys or the secret shared key of Alice and Bob, (*ii*) impersonate both Alice and Bob. In such a situation, Eve can intercept encrypted messages from Alice (Bob) to Bob (Alice), decrypts them with Alice's (Bob's) decryption key and re-encrypting them with her own encryption key before re-sending.

Furthermore, if Eve could, using a brute-force attack or some other method, obtain the private key of Alice or Bob (or their secret shared key), all of the above attacks could be easily performed. In a brute-force attack, Eve tests all possible valid keys to decrypt a cipher text of a known plain text in order to find out the correct key.

A.3 Auxiliary Services

An important issue of cryptographic services is whether they are supported by an infrastructure which provides a strong and secure set of auxiliary services such as generation, agreement, distribution and storage of cryptographic keys. Usually, key generation algorithms are based on random number generators. Public keys are usually distributed together with their digital certificates, which are packages of information attesting the ownership and validity of a cryptographic key. These certificates are usually signed by a trust third party, called a Certification Authority (CA).

A private or secret key must be kept protected from unauthorized copy and modification; this can be done in two ways: (*i*) it can be stored in a tamper-proof hardware, (*ii*) it can be stored in an encrypted and authentic form in general purpose hardware, such as random access memories, magnetic disks and tapes. This requires a Key Encryption Key (KEK for short) which, in turn, must be protected.

B Structure and Dynamics of Cryptographic Patterns

B.1 Information Secrecy Structure and Dynamics

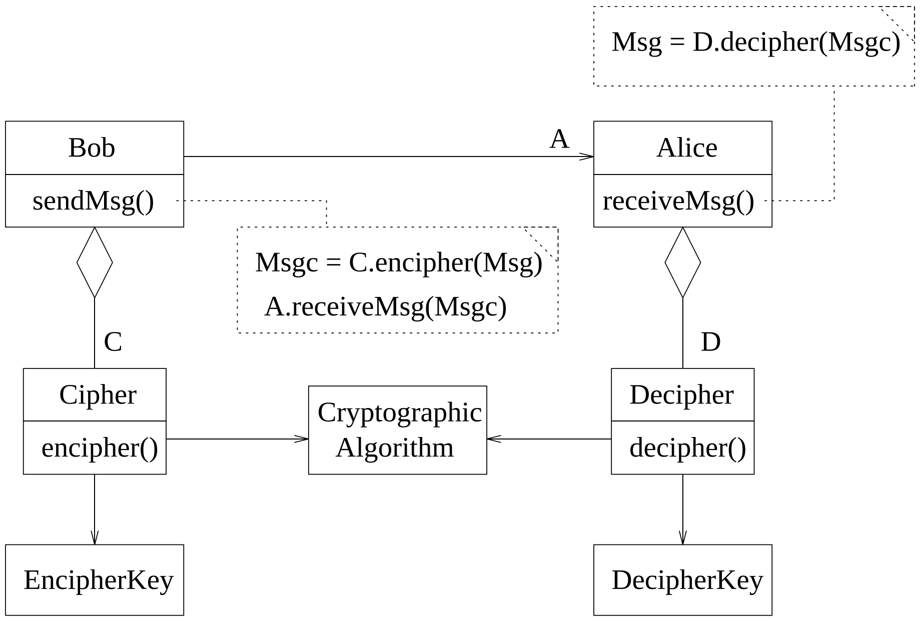

Figure 6: Information Secrecy Structure.

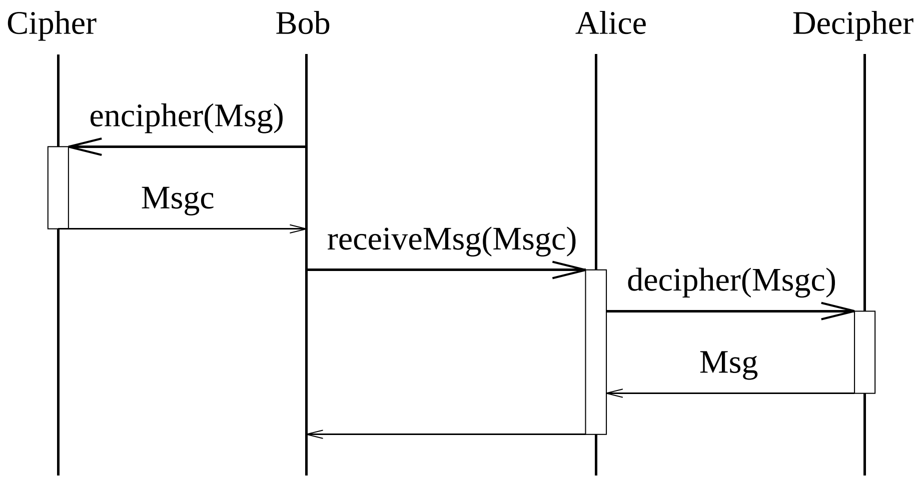

Figure 7: Information Secrecy Dynamics.

B.2 Message Integrity Structure and Dynamics

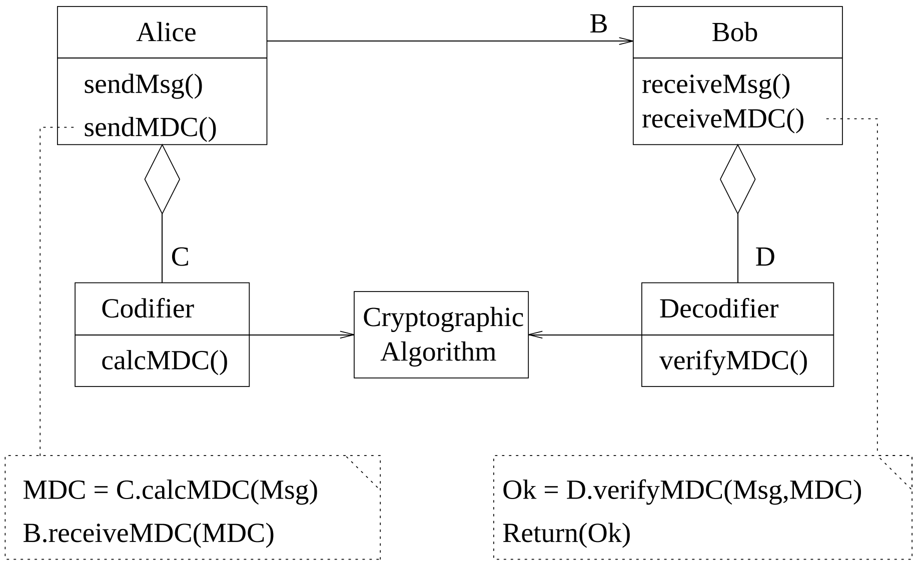

Figure 8: Message Integrity Structure.

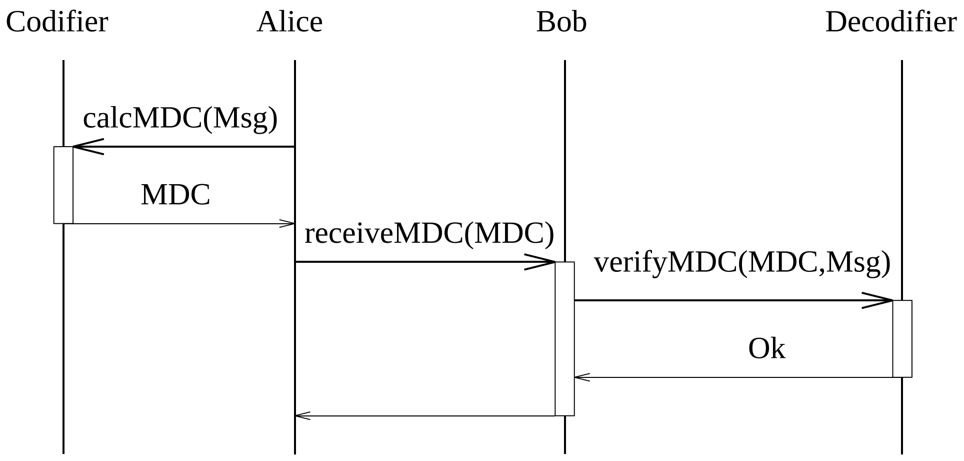

Figure 9: Message Integrity Dynamics.

B.3 Message Authentication Structure and Dynamics

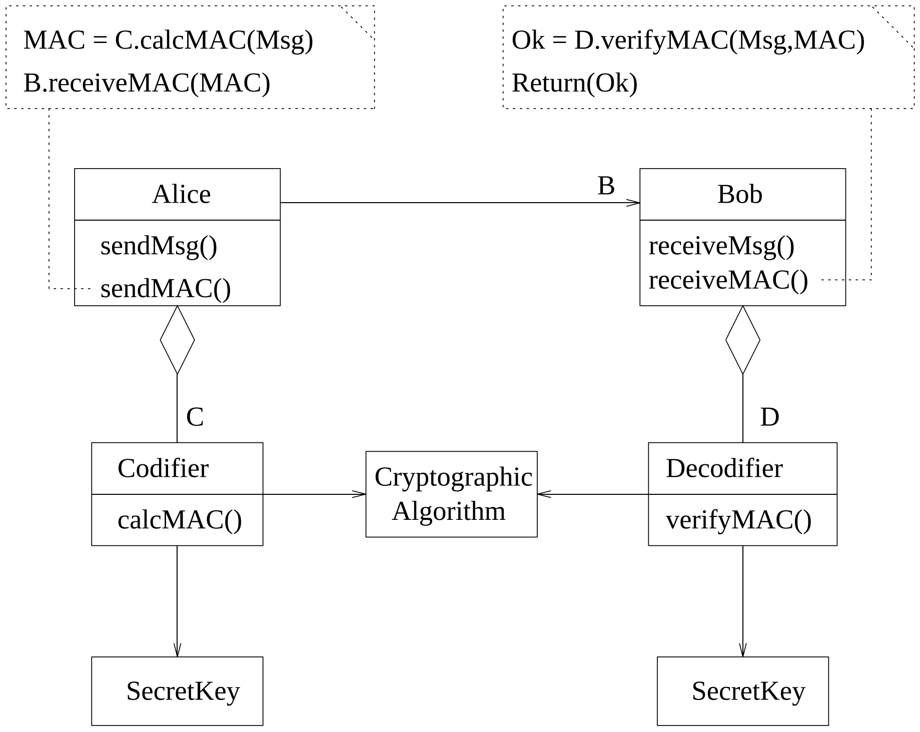

Figure 10: Message Authentication Structure.

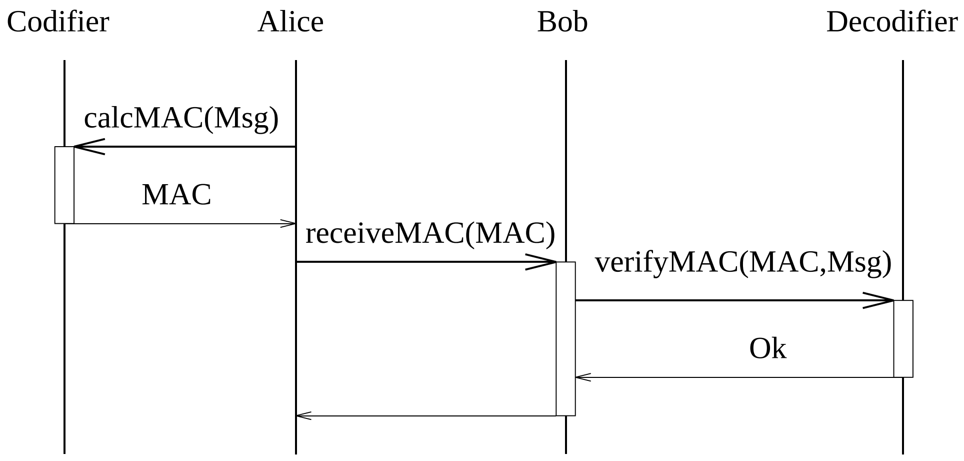

Figure 11: Message Authentication Dynamics.

B.4 Sender Authentication Structure and Dynamics

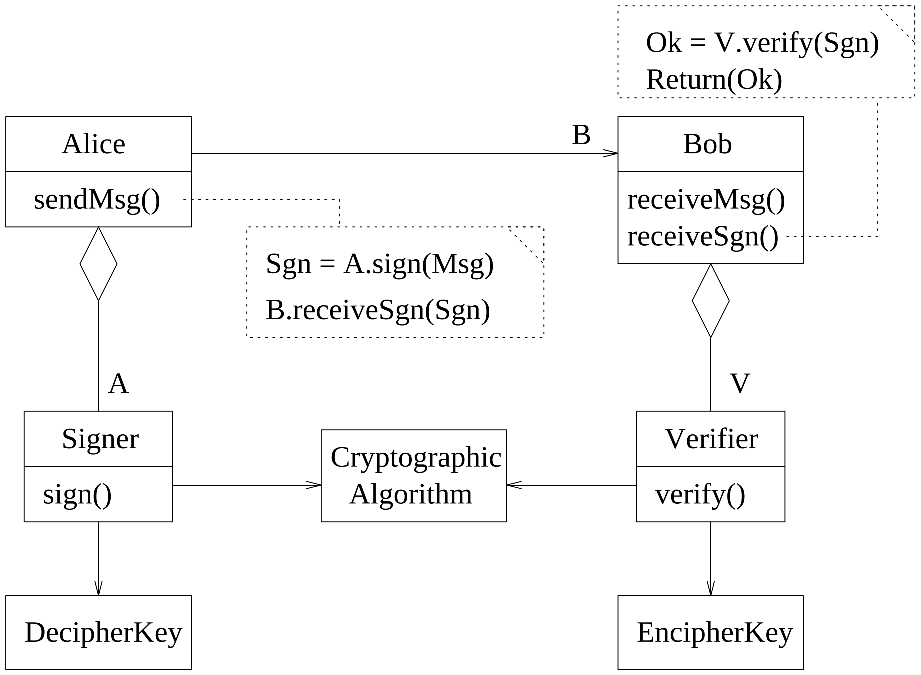

Figure 12: Sender Authentication Structure.

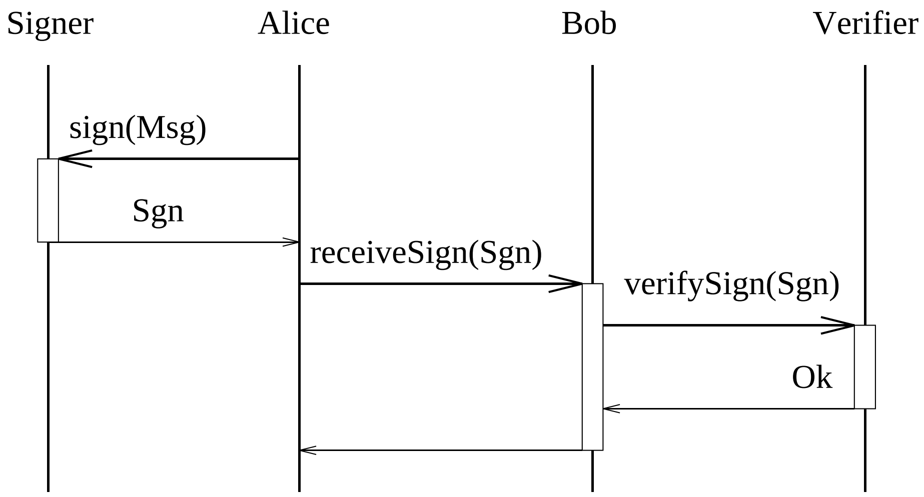

Figure 13: Sender Authentication Dynamics.

B.5 Secrecy with Authentication Structure and Dynamics

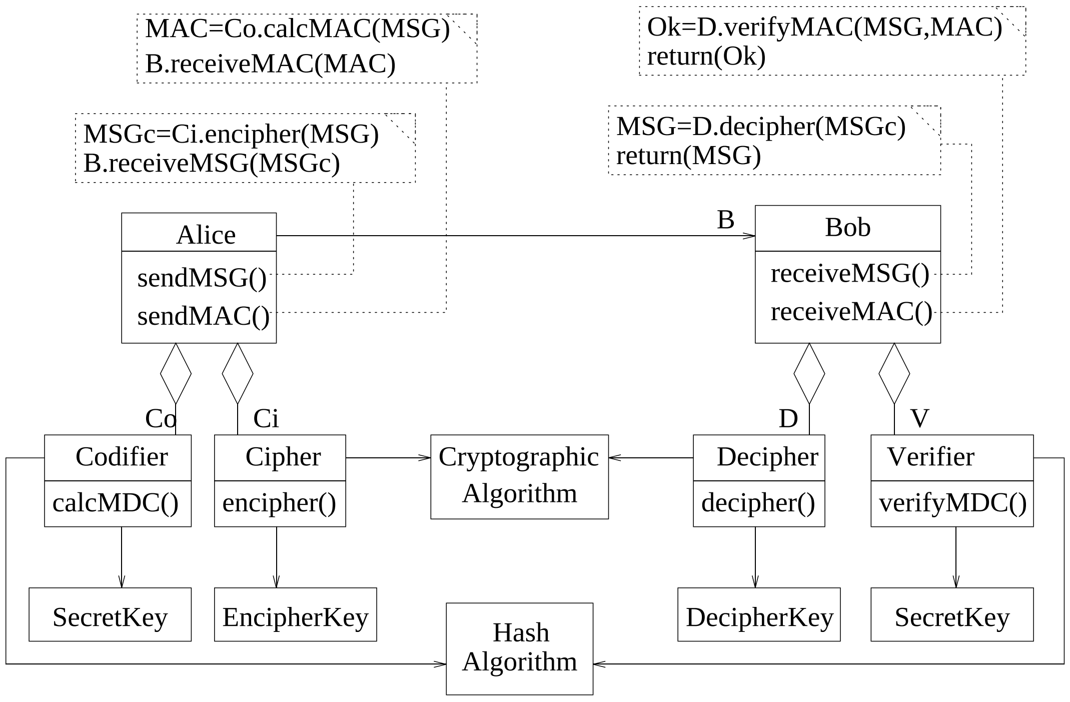

Figure 14: Secrecy with Authentication Structure.

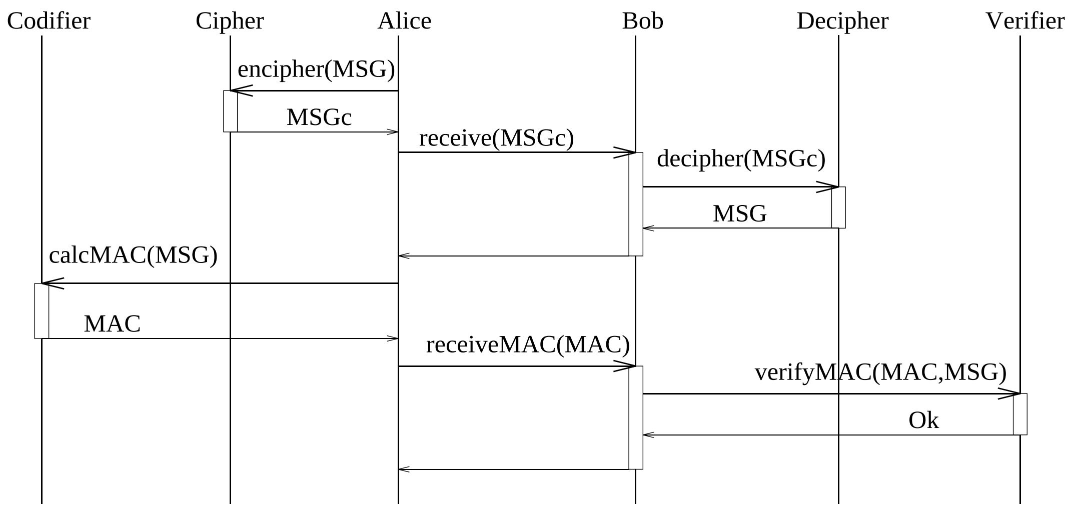

Figure 15: Secrecy with Authentication Dynamics.

B.6 Secrecy with Signature Structure and Dynamics

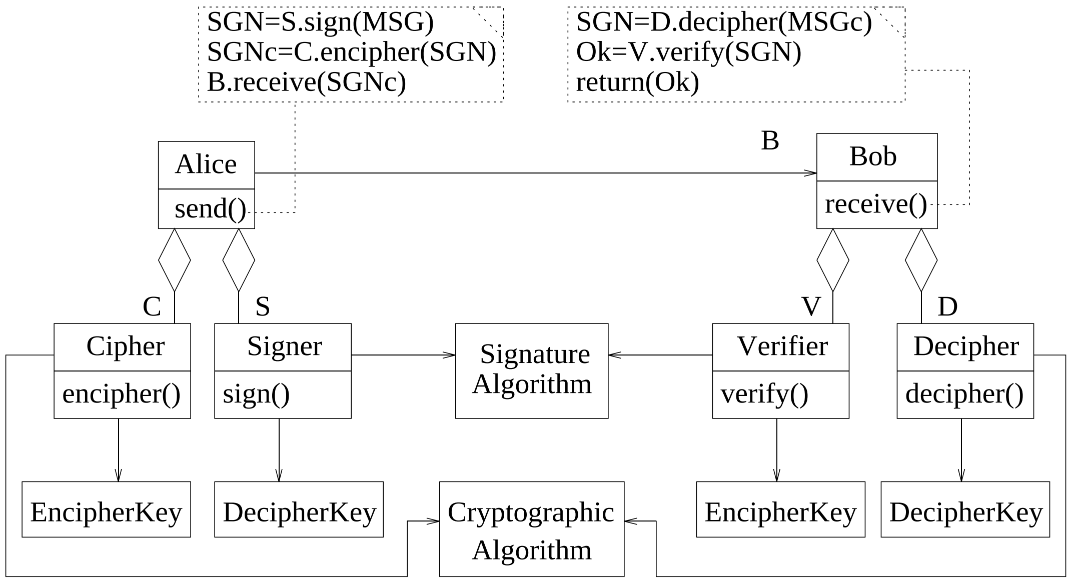

Figure 16: Secrecy with Signature Structure.

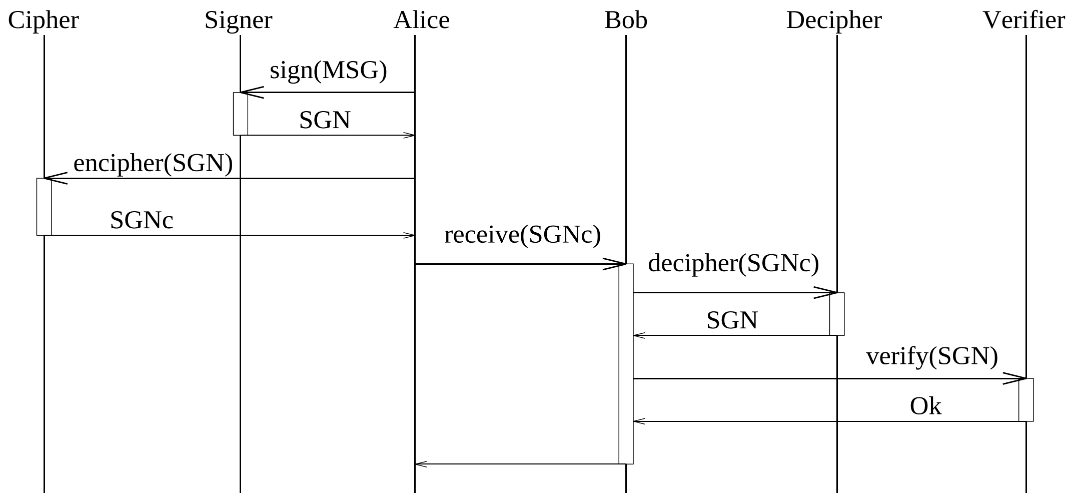

Figure 17: Secrecy with Signature Dynamics.

B.7 Secrecy with Integrity Structure and Dynamics

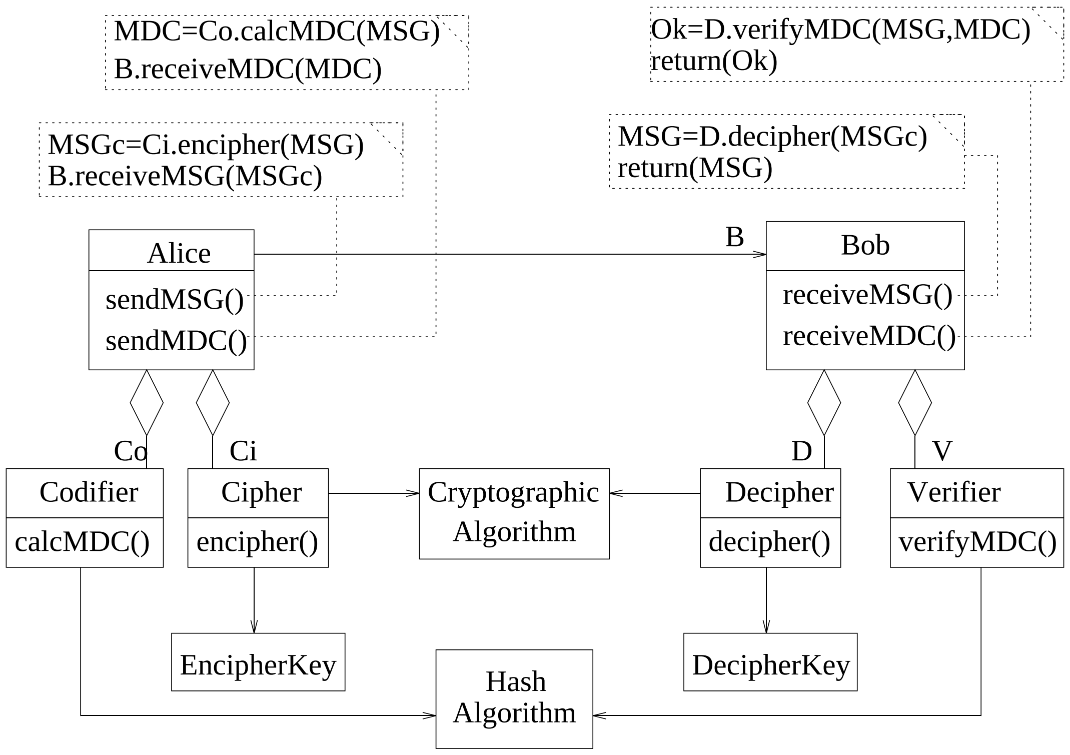

Figure 18: Secrecy with Integrity Structure.

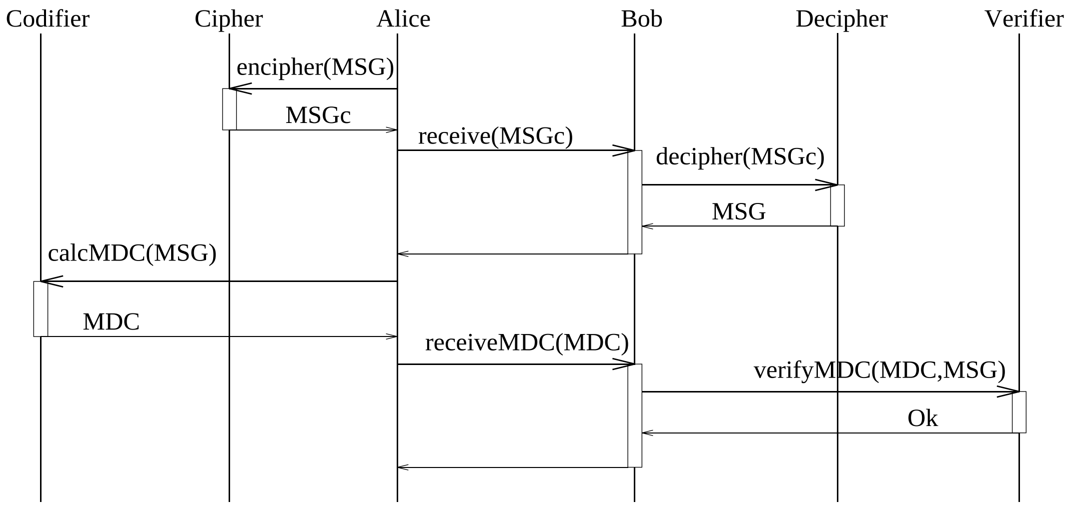

Figure 19: Secrecy with Integrity Dynamics.

B.8 Signature with Appendix Structure and Dynamics

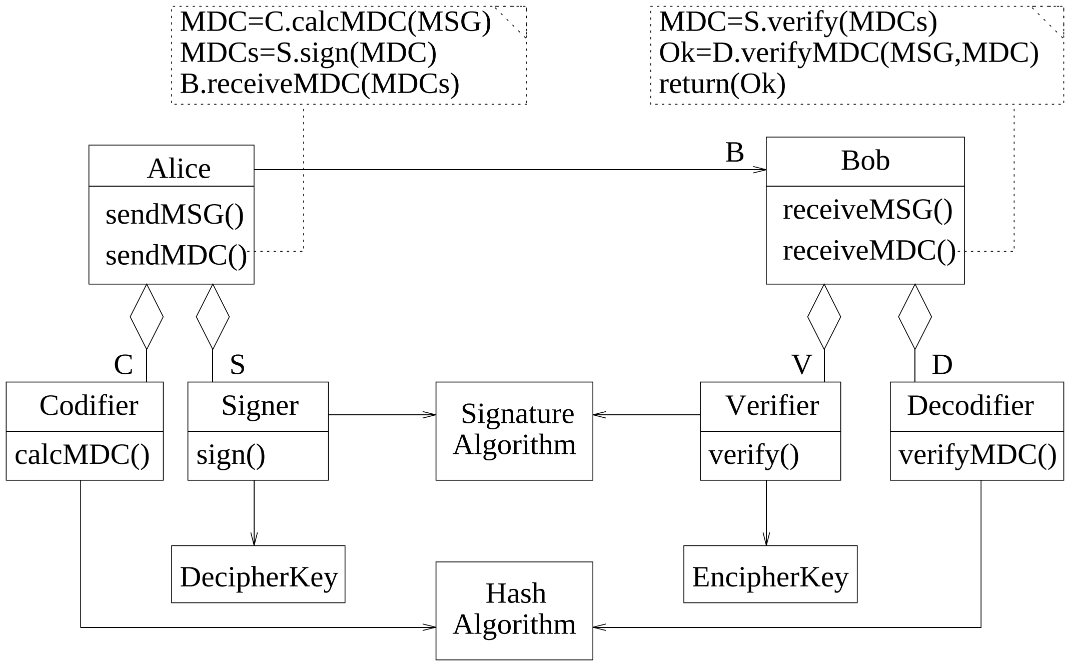

Figure 20: Signature with Appendix Structure.

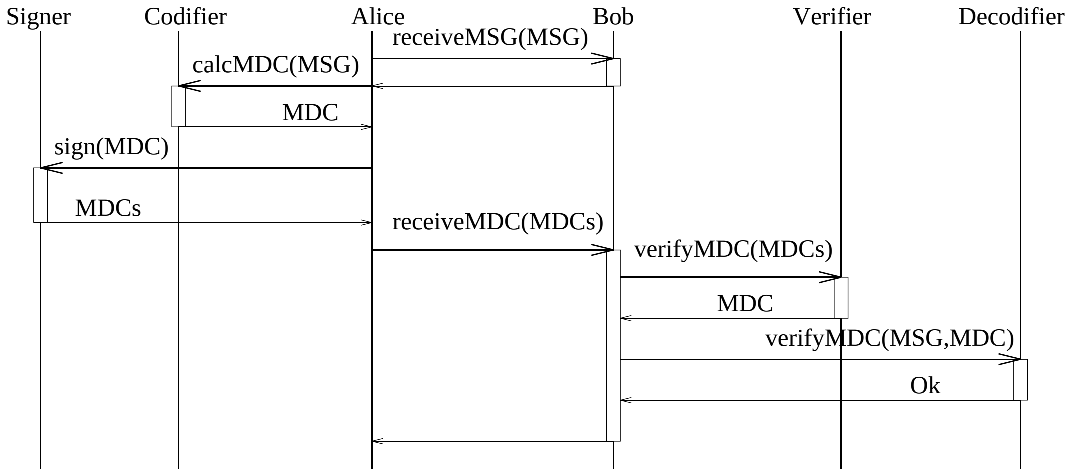

Figure 21: Signature with Appendix Dynamics.

B.9 Secrecy with Signature with Appendix Structure and Dynamics

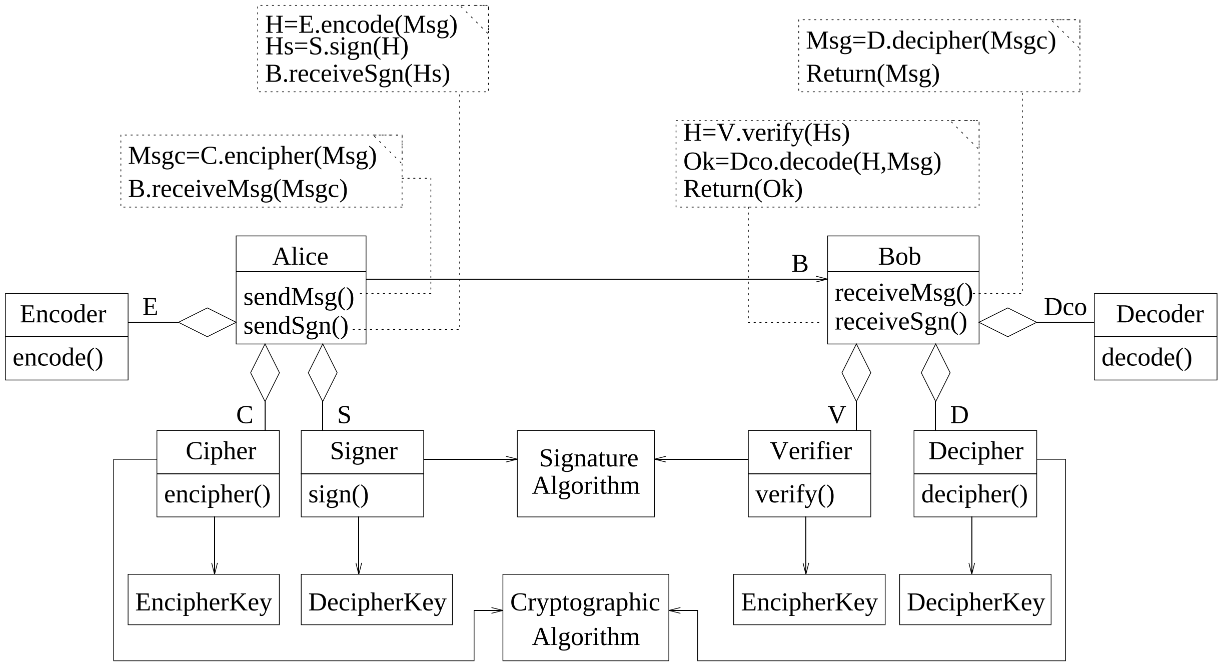

Figure 22: Secrecy with Signature with Appendix Structure.

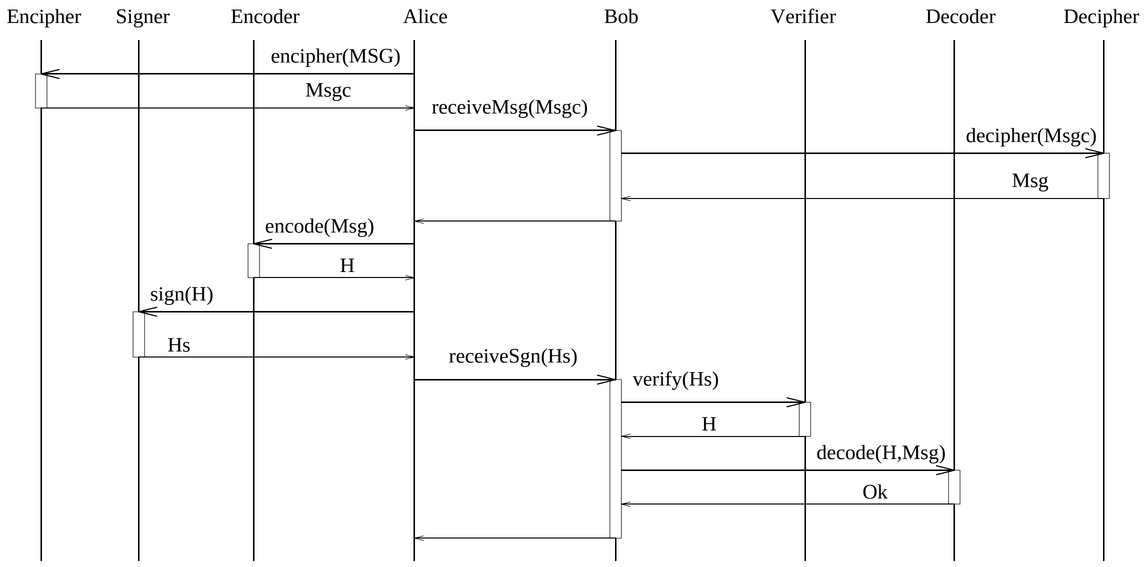

Figure 23: Secrecy with Signature with Appendix Dynamics.