Cryptographic Tools for Cloud Environments[[1]](#footnote-1)

James Alderman, Jason Crampton and Keith M. Martin

Information Security Group, Royal Holloway, University of London, Egham TW20 0EX, U.K.

**Abstract**

Cryptography provides techniques that can be used to implement core security services such as confidentiality and data integrity. We review some fundamental cryptographic mechanisms and identify some of the limitations of traditional cryptography with respect to cloud computing environments. We then review a number of relatively new cryptographic tools that have the potential to provide the extended security functionality required by some cloud computing applications.

Keywords:- Cryptography, Cloud Security, Encryption, Functional Encryption, Searchable Encryption, Homomorphic Encryption, Verifiable Computation

1. Introduction

Cloud computing provides several advantages to end users in terms of economical outsourcing of data storage and processing. However, in many practical settings client data may be sensitive in nature and should not be revealed to untrusted cloud service providers or transmitted in the clear over untrusted networks. Cryptography provides a mathematical toolkit of techniques for implementing core data security services. Traditionally, cryptography has focused on providing *confidentiality* and *integrity*: ensuring unauthorized entities cannot read data or modify data (without detection). However, modern cryptography can achieve significantly more functionality than this and has potential to provide effective solutions to specific security issues arising in cloud environments such as the enabling of (limited) processing of encrypted data. In this chapter, we identify limitations of traditional cryptographic tools, and then review a number of relatively recent cryptographic mechanisms that provide interesting functionality, all of which are potentially suitable for a wide range of cloud applications.

2. Fundamental Cryptographic Mechanisms

In this section, we review what can be considered as the fundamental tools of cryptography, including symmetric and public-key encryption, hash functions, message authentication codes and digital signatures. We discuss the functionality of these tools, as well as their limitations.

2.1 Symmetric Encryption

*Symmetric* or *private key* *encryption* is perhaps the most fundamental cryptographic mechanism and relies on a pre-shared key between the sender and the recipient of a message (or writer and reader, respectively). *Plaintext* messages (data objects) to be encrypted are elements of a *message space,* M, while the symmetric key shared between (at least) two entities is usually drawn uniformly at random from a *key space,* K. The space M is often regarded as the set of all (finite length) binary strings, while the size of K depends on a *security parameter,* which varies the strength of the encryption (a security parameter of 128 usually means that keys are drawn uniformly from the set of all 128-bit binary strings).

A symmetric encryption scheme comprises three algorithms. A *key generation* algorithm takes the security parameter as input and randomly selects a symmetric key *k* from K. The *Encrypt* algorithm is a randomized algorithm taking the symmetric key *k* and a message *m* from M as input, and outputting a ciphertext *c*. Finally, the *Decrypt* algorithm is deterministic and takes the symmetric key *k* and a ciphertext *c*, and returns the message *m*.

There are many notions of security for symmetric encryption. Perhaps the most common is *Indistinguishability against Chosen Plaintext Attacks (IND-CPA)*. Intuitively, this states that an adversary with the ability to observe the encryption of arbitrary messages may not distinguish which of two messages of its choice has been encrypted – that is, a ciphertext should hide all information about the encrypted plaintext. *Indistinguishability against Chosen Ciphertext Attacks (IND-CCA)* allows the adversary to also make decryption queries for ciphertexts of its choice. The choice of security notion depends on the context in which the scheme is used and what information an adversary is likely to observe in practice.

A symmetric encryption scheme that encrypts fixed size messages is often called a *block cipher* [1]. Longer messages may be split into fixed length blocks and chained together according to a *mode of operation* [2]. Particularly common modes of operation include *Cipher Block Chaining (CBC)* and *Counter Mode (CTR)*. The latter mode encrypts each block in parallel and is thus a good choice for encrypting particularly long plaintexts. The current standard for a block cipher is the *Advanced Encryption Standard (AES)* [3], which operates on blocks of 128 bits and supports key sizes of 128, 192 and 256 bits.

An alternative to a block cipher is a *stream cipher* [4], which generates a *key*  *stream* of pseudorandom bits, which may be prepared ahead of time, and then combines (simply XORs) this stream with the message on a bit-by-bit basis. In this way, arbitrary length messages may be accommodated as long as a stream of the appropriate length can be generated. Block ciphers in certain modes of operation (such as CTR) can act as stream ciphers, albeit with a potential loss in efficiency compared to dedicated stream ciphers, which can be very fast. One of the most commonly used stream ciphers is RC4 but this is known to have severe security weaknesses.

It should be noted that all symmetric encryption mechanisms have a requirement to agree secret keys in advance; hence a major challenge in all applications is to find a secure and efficient means of key distribution.

2.2 Public-key Encryption

*Public-key encryption* or *asymmetric* encryption [5], eases the problem of key distribution by removing the requirement for a pre-shared key. Instead, each entity A is associated with two keys: a *public key* used by an encryptor to send a message to A; and a *private (secret) key* used by A to decrypt ciphertexts that were encrypted using A’s public key. The public key may be transmitted in the clear (or published) so there is no requirement for a secure channel before transmitting a message (however, the public-key distribution channel should still be authenticated).

The public-key setting facilitates increased functionality as it is possible to encrypt objects *without* also having the ability to decrypt objects. It also allows a recipient to receive messages from multiple senders whilst keeping only a single private decryption key.

In general, public-key mechanisms are significantly slower than symmetric mechanisms. Thus, a *hybrid* model is often used whereby an object is (efficiently) encrypted using a symmetric encryption scheme, and the symmetric key itself is encrypted using a public-key scheme. Thus, the less efficient public-key scheme is only used to protect a reasonably short symmetric key, whilst the more efficient symmetric key scheme protects the larger data object.

Public-key encryption is formally defined in a similar way to symmetric encryption, with the key generation algorithm now outputting two keys. Security in the sense of IND-CPA and IND-CCA can also be defined for the public-key setting, where the adversary can form any ciphertext using just the public key.

2.3 Hash Functions

A *hash function* [6] is a compression function *H* that take an arbitrary length string and outputs a shorter string. For cryptographic applications we must ensure that the mapping *H* does not produce predictable collisions and that the precise mapping is hard to determine. The most important security properties of a hash function are:

*Collision resistance:* it should be computationally infeasible to find two distinct messages *x* and *z* such that *H(x) = H(z);*  that is, to find two messages that hash to the same value*;*

*Pre-image resistance:* given the result *y = H(x)* of *H* applied to a randomly chosen message *x*, it should be computationally infeasible to find a value *z* such that *H(z) = y*;

*Second pre-image resistance:* given a message *x* it should be computationally infeasible to find another message *z* such that *H(x) = H(z)*.

Hash functions find numerous uses in cryptography, from dedicated applications such as password protection through to acting as components in more complex cryptographic tools such as digital signature schemes.

2.4 Message Authentication Codes

A *Message Authentication Code (MAC)* [7]detects any modification of a message. As with symmetric encryption, the sender and receiver of the message must pre-share a secret key *k*. The sender computes a *tag* *t* using the message and the secret key. The tag is transmitted alongside the message, and the receiver verifies the MAC using the message, tag and secret key.

Formal MAC security is captured by *existential unforgeability under chosen-message attack (EUF-CMA),* which requires it to be computationally infeasible to generate a valid looking tag for a new message, for which no tag has previously been seen, without access to the key *k*.

2.5 Digital Signature Schemes

*Digital Signature Schemes* [8] are public-key analogues of MACs, which preserve the integrity of messages without requiring a pre-shared secret. The key generation algorithm outputs both a private signing key, kept by the sender, and a verification key, which is published. The signing algorithm takes the secret signing key and the message, and produces a signature. The verification algorithm takes the message, signature and public verification key and accepts the signature if the message has not been altered. As with MACs, we define a notion of security known as *existential unforgeability under chosen-message attack (EUF-CMA),* whereby an adversary given the verification key and the ability to request signatures on arbitrary messages should not be able to generate a valid signature on a message that has not previously been signed.

As with encryption, the asymmetric setting eases key distribution, particularly when communicating with multiple parties (a single signature can be verified by multiple parties using the published verification key), but this comes at the expense of computational efficiency. Signatures are *publicly verifiable*, meaning that if one receiver verifies the signature correctly then it can be assumed that all other recipients may do the same; clearly this may not be true in the case of MACs where each recipient holds a different verification key and a MAC is created per key. Publishing the verification key also means that signatures are *transferable*; that is, given a message and a valid signature, a third party can verify the signature even if not the intended recipient. A final useful property of digital signatures is *non-repudiation,* which means that the signer of a message cannot deny having done so. MACs cannot provide such functionality since the key is shared between the signer and verifier only; it is not possible for a third party to verify that the signer held the particular signing key used. More advanced forms of signatures allow sets of users to jointly sign messages, signatures to be generated without the signer knowing the contents of the message, and for authorized users to modify designated portions of the message.

2.6 Authenticated Encryption

*Authenticated encryption* [9] combines the confidentiality properties of encryption with the integrity properties of MACs and digital signatures, thus assuring that a received message has not been read by unauthorized entities and has not been altered since its creation by the sender. Integrated methods to achieve this in the symmetric setting either combine a block cipher with a MAC or use a special authenticated mode of operation for a block cipher (such as GCM or CCM). Some modes allow for additional data to be authenticated but not encrypted. The public-key analogue of an authenticated mode of operation is known as *signcryption* [10, 11], which integrates asymmetric encryption and digital signature schemes. Authenticated encryption schemes are typically more efficient than manual combinations of separate privacy and integrity mechanisms.

3. Limitations of Conventional Cryptography

Cloud environments provide several challenges that are not addressed by conventional cryptographic mechanisms. Three of the main limitations of conventional cryptography when applied to cloud settings are as follows:

*Inability to conduct processing on encrypted data.* Conventional cryptographic mechanisms can be used to protect the confidentiality and integrity of stored and transmitted data. However, a natural requirement in the cloud is for a *cloud service provider* (CSP) who stores encrypted data to *process* data on behalf of a client. Without access to the data itself, it is hard for a CSP to perform meaningful processing, especially as conventional cryptography often requires it to be *hard* to meaningfully manipulate encrypted data. An inefficient solution would be to return the encrypted data to the client for local processing. However, some modern cryptographic tools (which we will discuss later) permit some computation directly over encrypted data.

*Incorporation of data access policies*. Conventional cryptography typically operates in a point-to-point setting where the sender knows the intended recipient, be that through a pre-shared key or through an associated public key. In cloud settings, however, it is likely that a dynamic set of clients may communicate with a CSP. It may be infeasible to compute a ciphertext for each potential user of a piece of encrypted data. Indeed, it may not even be possible to *define* (in terms of individual identities) the entities that should be given the capability to interact with particular encrypted resources. Moreover, in conventional systems, data is often stored in a single location within a trusted zone, with a trusted reference monitor enforcing access control to protected resources. In cloud environments, data may be outsourced to multiple locations over which the data owner exerts no control, making access to resources more problematic. As we will shortly discuss, some modern encryption schemes include built-in access control mechanisms that allow decryption policies to be enforced remotely on behalf of the data owner.

*Reliability of the encrypted data holder.* Most conventional cryptography, particularly symmetric cryptography, relies on a degree of trust between the communicating entities. However, in a cloud environment a CSP typically lies outside of the trusted domain of the clients. In particular, a CSP could make unintentional errors, especially when asked to process encrypted data. In extremis, a malicious CSP could attempt to preserve resources by returning incomplete results or even deleting some outsourced data. We will discuss some new cryptographic mechanisms that reduce the necessary level of trust held by a client with respect to the holder of encrypted data.

One additional problem created by cloud environments is that operational requests made for the processing of encrypted data could be made over public channels and are certainly visible to the potentially untrusted CSP. A further problem is that optimization of storage costs is difficult when CSPs are not aware of raw data that is transmitted to them in encrypted form (in conventional cryptography, multiple ciphertexts encrypting the same data using different keys should appear completely unrelated). We will also discuss some tools for addressing both of these issues.

4. Cryptographic Mechanisms for the Cloud

We now discuss a range of relatively new cryptographic mechanisms that address some of the limitations of conventional cryptography and have potential for deployment in cloud environments.

4.1 Processing Encrypted Data

We first consider methods for performing specific computations on encrypted data, which could prove useful in cloud environments.

4.1.1 Searching over Encrypted Data

One of the most basic processing tasks that might be required to be performed on encrypted data is to perform keyword search. However, once data has been encrypted, this operation becomes extremely difficult because conventional ciphertexts should not leak any information about the underlying plaintext. Additionally, queries themselves may need to be encrypted as they may reveal information about the data being searched for.

*Searchable encryption* (SE) schemes are encryption schemes designed to address this problem by encrypting data alongside special indices that permit a limited search capability. A range of SE schemes have been proposed, varying in the expressiveness of queries and the degree of privacy offered. Some SE schemes [12, 13] return all documents containing a *single* keyword, whilst other solutions [14] allow conjunctive searches for documents containing a set of keywords, or allow for general Boolean formulae [15], range and subset queries [16] or even SQL queries [17]. *Fuzzy* searches [18] seek words that are “close” to a given keyword. Security for SE schemes considers *data confidentiality* and *query confidentiality*, as well as *search pattern privacy* (so that an adversary that sees two queries with the same output cannot tell whether the queries were identical), and *access pattern privacy* (so that an adversary cannot learn the result of queries).

Early SE schemes allow a single data owner to issue queries. Subsequent work [19] enables many clients to write to a database by encrypting data segments with the public key of a single user who may form searches using the corresponding private key. Other solutions allow a single data owner to grant and revoke the ability to search their files [12], or combine both properties to allow multiple readers *and* multiple writers. In terms of efficiency, some schemes [12] include a search phase in which the workload of the server is not linear in the number of uploaded documents but rather in the number of documents that match the query.

4.1.2 Homomorphic Encryption

By default, traditional encryption schemes do not permit meaningful combinations of ciphertexts. However, some encryption schemes are *homomorphic* in nature and allow some computations to be performed on encrypted data. Certain operations can be applied to two ciphertexts such that the result, when decrypted, produces a plaintext as if the operation had been applied to the plaintexts themselves. For example, let *C1* be the encryption of a message *m1* and let C2encrypt *m2*. Then, for example, in certain *homomorphic encryption schemes*, multiplying *C1* and *C2* together will produce a new ciphertext C3 which will decrypt to reveal a plaintext equal to *m1* times *m2*. It should be noted that, by design, homomorphic encryption schemes are *malleable* (ciphertexts can be altered and remain valid).

Schemes that exhibit homomorphic properties for a *specific* operation are known as *partially homomorphic encryption schemes*. On the other hand, if the set of permissible operations enables arbitrary computations to be performed then the schemes are referred to as *fully homomorphic* [20]. Such schemes are very powerful since they allow arbitrary computation on encrypted data, and thus potentially fit the cloud setting particularly well as the untrusted CSP never requires access to the plaintext data. Unfortunately, current schemes (referred to as *somewhat homomorphic*) tend to be limited in the number of operations that may be applied before decryption will no longer succeed, or are inefficient in terms of speed or the size of parameters and ciphertexts.

4.1.3 Computing Aggregates over Encrypted Data

Since fully homomorphic encryption is not yet practical for general deployment, several cryptographic mechanisms have been designed for more specific uses. One such example is *privacy preserving data aggregation,* which allows specific types of computation to be performed (generally the sum) on encrypted data. The data is assumed to come from multiple independent sources, which are reluctant to share their sensitive information with either other sources or the aggregator. This has led to an active area of research where proposed solutions mainly rely on homomorphic encryption and secret sharing techniques. Some schemes achieve *aggregator obliviousness* using a trusted dealer that provides the aggregator with the sum of users' secret keys, which in turn allows the decryption of the sum of users' data. Other schemes handle dynamic user populations and arbitrary user failures. Recently, Leontiadis et al. [21] removed the need for trusted key dealers while supporting dynamic group management and user failures.

4.1.4 Order Preserving Encryption

*Order preserving encryption* [22–24]allows a CSP to perform *range queries* on encrypted data in order to return relevant results to a client query. This is a form of deterministic symmetric encryption where numerical comparison operators can be applied to encrypted numerical data. It has natural applications to querying encrypted databases. The scheme of Boldyreva *et al.* [23] claims to achieve such numerical range searches in logarithmic time (in the size of the database).

4.2 Functional Encryption

*Functional encryption* extends traditional public-key encryption to allow the holder of a private key to learn a specific function of an encrypted message, but nothing else. This function could return the message itself (as for traditional public key encryption), return the message only if some additional criteria is met, or may produce the output of some computation specified by the message and private key. In the context of cloud, functional encryption can be deployed as a cryptographic enforcement mechanism for access control policies. Functional encryption allows the encryptor (client) to specify an access control policy in terms of identities or more general descriptive attributes; decryptors may access the data if and only if they satisfy this policy. Thus, data owners retain control of which entities may learn their data without requiring explicit prior knowledge of users.

There are several specific types of functional encryption scheme that could be of interest in cloud environments.

4.2.1 Identity-based Encryption

*Identity-based Encryption (IBE)* [25] allows encryptors to specify an arbitrary identity string (user name, email address, IP address *etc*.) whilst preparing a ciphertext, rather than using a pre-defined public key. A decryptor can request (either beforehand or subsequently) a decryption key associated with an identity from a key generation authority (usually after proving authorization for the identity). The plaintext is successfully recovered if and only if the identity associated with the ciphertext matches that associated with the key. Since identity strings can be arbitrary, it is possible to append the current day, for example, in order to specify a lifetime for a decryption key. Similarly, one could append access rights or different separations of duty, *etc.*

4.2.2 Attribute-based Encryption

*Attribute-based Encryption* (ABE) is useful when the authorized set of decryptors cannot easily be stated explicitly in terms of identifier strings (*e.g.* because the user population is too large or changes too frequently). Instead, authorized decryptors can be described in terms of *attributes*. ABE comes in several variants that vary based on the form of the key and ciphertexts. In *Key-Policy ABE (KP-ABE)* [26], the ciphertext contains a set of attributes that describe the classification and contents of the plaintext, whilst the decryption key is associated with an *access structure* (which describes the access policy). Decryption succeeds if and only if the set of attributes satisfies the access structure. Thus, a user can be issued a key for a formula specifying their access rights (*e.g.* Manager OR (ClearanceLevel2 AND Accounts)), while ciphertexts can be associated with a set of attributes describing its contents or required level of protection (*e.g.* {YearlyReport, Accounts, ClearanceLevel2}). On the other hand, *Ciphertext-Policy ABE* (CP-ABE) [27] reverses the association of attribute sets and access structures. Ciphertexts are now formed with an associated formula over attributes (describing the users that should be able to read the contents) whilst decryption keys are issued for an attribute set.

4.2.3 Predicate Encryption

*Predicate Encryption* (PE) [28] generalizes the previous notions of functional encryption, particularly KP-ABE. Decryption keys are associated with a *predicate F* over attributes and ciphertexts are associated with a set of attributes *I*. Decryption succeeds if and only if *F*(*I*) *=* 1. Thus, if *F*(*I*) *=* 0 then no information is learnt about the encrypted message; this property is referred to as *payload hiding*. Furthermore, some schemes can achieve a stronger notion of *attribute hiding* whereby, as well as hiding the message, no information is learnt about the attribute set *I* beyond what is naturally leaked by the decryption functionality – that is, the result of *F*(*I*). Many PE schemes focus on the specific predicates of inner products, which have been shown to encompass useful functionality such as Boolean formulas in conjunctive normal form and disjunctive normal form, threshold policies and polynomial evaluation.

4.3 Verifiable Computing

It is commonly suggested that CSPs should be “honest-but-curious”, generally meaning that they are trusted to follow the rules of any process but cannot be fully trusted with respect to privacy of data that they happen to observe. However it is not necessarily always the case that such a level of trust can be placed in a CSP. Several new cryptographic mechanisms provide services that may be appropriate in cloud environments with reduced levels of trust in CSPs.

**4.3.1 Verifiable Outsourced Computation**

One concern arises in environments where a CSP is not trusted to return the correct result of a processing computation. In *verifiable outsourced computation* (VC), a client delegates the execution of computationally demanding operations to the cloud and receives the results alongside a cryptographic proof of their integrity. These proofs allow the detection of any server misbehavior and, at the same time, do not let a client falsely accuse a server of misbehaving.

Features of verifiable computation schemes include *public verifiability*, which ensures that *anybody* can verify the correct execution of outsourced operations using only public information. Most VC schemes use *non-interactive proofs,* which restrict the necessary level of interaction between provers and verifiers. Many different techniques have been used to build VC schemes, including fully homomorphic encryption [29, 30] and KP-ABE [31]. Pinocchio [32] applies *succinct non-interactive arguments of knowledge* (SNARKs) to achieve public verifiable computation of arbitrary functions.

**4.3.2 Verifiable Storage**

Another concern arises when CSPs are not trusted to preserve the integrity of outsourced data in their charge. A simple solution is for the client to compute and store a checksum (such as a MAC) of the data. However, this kind of verification scales poorly in cloud environments where huge amounts of data are stored. *Verifiable storage* schemes aim to make verification more efficient than downloading the entire data set, and allow clients to perform integrity checks as many times as needed. Solutions broadly fall into two categories: (i) *deterministic solutions* that offer an undeniable guarantee of integrity, and (ii) *probabilistic solutions* in which the verifier is convinced of the integrity of the data with a certain probability only. Deterministic solutions incur considerable computation and communication complexity, generally linear in the size of the entire data. Most schemes propose probabilistic optimizations based on random sampling; instead of checking the entire file, these proposals check the integrity of a subset of segments included in the file. Probabilistic solutions can generally be classed as either *Provable Data Possession* (PDP) (verifying that the data is held by the server) [33] or *Proofs of Retrievability* (POR) (verifying that data is recoverable from the server, even if small modifications have been made by a malicious server) [34]. Early proposals [33] deal with static data in the context of archival or back-up storage, whilst others allow for efficient updates of the data (modification, deletion or insertion of blocks) and for efficient integrity verification to ensure that the server stores the latest version of the outsourced data. Moreover, some solutions allow verification to be delegated to a third-party auditor [35] and to render this public verification privacy-preserving [36].

4.4 Other Tools

In addition to the three classes of cryptographic mechanism just discussed, there are several other relatively recent cryptographic tools that have the potential for deployment in cloud environments.

**4.4.1 Proxy re-encryption**

*Proxy re-encryption* [37]allows a semi-trusted intermediary (*proxy*) to convert a ciphertext intended to be read by one entity into one that can be read by another, without the proxy decrypting the ciphertext, or otherwise learning the message itself. One example application is to manage access to encrypted data stored on a cloud server, which acts as the proxy. The stored ciphertexts can be transformed such that they can be decrypted by authorized entities, yet the server itself remains unable to read the data.

**4.4.2 Oblivious RAM**

If a CSP is untrusted, a client may wish to access data and for the CSP to process stored encrypted data without revealing which data items are being used, how frequently they are accessed, and in what order. *Oblivious RAM (ORAM)* [38] aims to hide the memory locations accessed by RAM programs. Clients need only store a small amount of data, whilst servers store *O(n)* data for *n* outsourced data items, and each access request can be replaced by *O(log2n)* accesses [38]. Data must be stored in encrypted form and associated with an index. In order to hide the access patterns, further *dummy* accesses are made in other locations. If read and write requests should also look equivalent then each read operation must include at least one dummy write operation to the same location, and *vice versa*. The location that data is stored in must be independent of the index and two accesses to the same index should not necessarily access the same location. In general, ORAM schemes work by imposing some additional structure on the memory and then performing a read (or write) operation to a set of locations, using a combination of sorting and hashing algorithms. Since only the client knows which of these accesses was the desired one, no information is revealed to the CSP.

**4.4.3 Format Preserving Encryption**

*Format Preserving Encryption* (FPE)[39] enables formatted data to be symmetrically encrypted to ciphertexts that conform to the same formatting rules (*e.g.* credit card numbers are encrypted to random, valid credit card numbers). The encryption induces a pseudo-random permutation over all validly formatted strings. This property can be useful for storing encrypted databases where data fields must follow specific formatting rules. In particular, it is useful when upgrading legacy outsourced database solutions to be secure; in general, it is not possible to simply encrypt the data using a non-format preserving encryption scheme without changing the structure of the database itself. One way to achieve FPE is the *rank-then-encipher* approach where the set of all valid strings are numbered according to some ranking function. Then, using a simpler integer FPE scheme (which encrypts integers to integers), one can encrypt the rank of the message. The produced ciphertext will be an integer that indexes some random (correctly formatted) message from the message space, which forms the final ciphertext. Currently, such ranking functions exist for all formatted domains where the format can be expressed by a regular language.

**4.4.4 Secure Deduplication**

*Secure Deduplication* [40] provides a space-efficient storage solution for outsourced data. If two users request to store the same data file, a cloud server may wish to save storage costs by only storing one copy of the file for both users. However, this is difficult if data is encrypted prior to being outsourced, as the security properties of a (randomized) encryption scheme will result in the ciphertexts for both files appearing entirely unrelated and, furthermore, storing just one ciphertext will prevent other users from recovering the data without holding the same decryption key. One solution to this problem is to use *message-locked encryption* (MLE) [40], which encrypts the message under a key derived deterministically from the message itself. For example, the key could be defined to be the result of a hash function applied to the message. A *tag* may also be generated that the server may use to detect duplicates. Privacy holds only when the message space has sufficient min-entropy. Another important security property is known as *tag consistency,* which requires that it is hard to force an honest client to recover a message different from that which it uploaded (*e.g.* by forging a tag such that the server believes the encryption of two different messages represent the same message and therefore deletes the second copy and returns the message).

5. Closing Remarks

In this chapter, we have briefly surveyed a number of relatively recent cryptographic mechanisms that have potential uses in security applications within cloud environments. While these tools show great promise in overcoming some limitations of conventional cryptography, it is important to apply some words of caution before recommending their immediate application.

While some of the discussed cryptographic tools are beginning to see deployment in cloud environments (such as searchable encryption and some functional encryption schemes), many others are still relatively young and the relevant theory is still under development. Several tools are only assured to be secure within the context of highly specific security models. As such, security levels may not yet be at acceptable levels.

Furthermore, cryptography is not (yet) able to *efficiently* provide all desirable functionality, especially when it comes to the processing of outsourced, encrypted data. For example, whilst fully homomorphic encryption remains promising to enable arbitrary computations on encrypted data, finding a truly practical, general-purpose scheme remains an open research area. In general, many of the tools discussed in this chapter are probably not yet efficient enough for practical deployment.

Cryptography is an area that has traditionally proved most effective when informed communities agree on the best available techniques. This is partly because of the difficulty of identifying effective mechanisms, but also because many applications benefit from compatibility. As yet, appropriate standardization activities are at relatively early stages for many of the mechanisms discussed in this chapter.

Nonetheless, the tools discussed in this chapter represent exciting developments in the theory of cryptography. We fully expect refinements and improvements to occur over the coming years that will result in these tools becoming effective practical mechanisms for securing data in cloud environments, and indeed elsewhere.

Review Questions

1. In general, what fundamental role does cryptography play in providing security for a computer system, whether in a cloud environment or otherwise?
2. What are the main limitations of traditional cryptography with respect to the security of typical cloud environments?
3. What types of processing operations is it possible to do on encrypted data using the tools described in this chapter?
4. How might a typical cloud environment benefit from the deployment of attribute-based encryption?
5. Searching over encrypted data is a potentially useful process to be able to conduct in a cloud environment, but what problems might arise from doing this over an insecure communication channel to an untrusted cloud server?
6. While the mechanisms described in this chapter appear to offer great promise, why should we be cautious about seeking to deploy them today?

**Acknowledgments**

This chapter was adapted from a review conducted as part of the European Union project H2020-644024 "CLARUS". We would like to thank the following researchers who contributed to the initial deliverable: Monir Azraoui, Aida Calviño, Josep Domingo-Ferrer, Melek Önen, David Sánchez, Cédric Van Rompay, and Oriol Farràs Ventura.

References

1. ISO/IEC 18033-3:2010 Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers, (2010).

2. Dworkin, M.: Recommendation for block cipher modes of operation. methods and techniques. National Institute of Standards and Technology Special Publication. (2001).

3. National Institute of Standards and Technology: 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication. 197, 311–441 (2001).

4. ISO/IEC 18033-4:2011, Information technology — Security techniques — Encryption Algorithms — Part 4: Stream Ciphers, 2nd edition, (2011).

5. ISO/IEC 18033-2:2006 Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers, (2006).

6. Dang, Q.: Recommendation for applications using approved hash algorithms. National Institute of Standards and Technology Special Publication. 107, (2008).

7. ISO/IEC 9797-1:2011 Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher, (2011).

8. National Institute of Standards and Technology: Digital Signature Standard (DSS). Federal Information Processing Standards Publication. 186-2, 1–119 (2009).

9. ISO/IEC 19772:2009 Information technology — Security techniques — Authenticated encryption, (2009).

10. Yung, M., Dent, A.W., Zheng, Y.: Practical Signcryption. Springer Science & Business Media (2010).

11. ISO/IEC 29150:2011 Information technology — Security techniques — Signcryption, (2011).

12. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security. 19, 895–934 (2011).

13. Bellare, M., Boldyreva, A., O’Neill, A.: Deterministic and efficiently searchable encryption. Advances in Cryptology - CRYPTO 2007. LNCS 4622, 535–552 (2007).

14. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. Applied Cryptography and Network Security. LNCS 3089, 31–45 (2004).

15. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. Advances in Cryptology – CRYPTO 2013. LNCS 8042, 353–373 (2013).

16. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. Theory of Cryptography. LNCS 4392, 535–554 (2007).

17. Popa, R., Redfield, C.: CryptDB: protecting confidentiality with encrypted query processing. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP ’11, 85–100 (2011).

18. Li, J.L.J., Wang, Q.W.Q., Wang, C.W.C., Cao, N.C.N., Ren, K.R.K., Lou, W.L.W.: Fuzzy keyword search over encrypted data in cloud computing. INFOCOM, 2010 Proceedings IEEE. 1–5 (2010).

19. Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith III, W.E.: Public key encryption that allows PIR queries. Advances in Cryptology – CRYPTO 2007. LNCS 4622, 50–67 (2007).

20. Gentry, C.: A fully homomorphic encryption scheme, (2009).

21. Leontiadis, I., Elkhyaoui, K., Molva, R.: Private and dynamic time-series data aggregation with trust relaxation. Cryptology and Network Security. LNCS 8813, 305–320 (2014).

22. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. SIGMOD ’04, 563–574 (2004).

23. Boldyreva, A., Chenette, N., Lee, Y., O’Neill, A.: Order-preserving symmetric encryption. Advances in Cryptology – EUROCRYPT 2009. LNCS 5479, 224–241 (2009).

24. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. Proceedings - International Conference on Distributed Computing Systems. 253–262 (2010).

25. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. Advances in Cryptology - CRYPTO 2001. LNCS 2139, 213–229 (2001).

26. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., and di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006).

27. Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. Public Key Cryptography – PKC 2011. LNCS 6571, 53–70 (2011).

28. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Advances in Cryptology – EUROCRYPT 2008. pp. 146–162. Springer (2008).

29. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. Advances in Cryptology – CRYPTO 2010. LNCS 6223, 483–501 (2010).

30. Barbosa, M., Farshim, P.: Delegatable homomorphic encryption with applications to secure outsourcing of computation. Topics in Cryptology – CT-RSA 2012. LNCS 7178, 296–312 (2012).

31. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. Theory of Cryptography. LNCS 7194, 422–439 (2012).

32. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy, pp. 238–252 (2013).

33. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. Proceedings of the 14th ACM Conference on Computer and Communications Security, 598–609 (2007).

34. Juels, A., Kaliski Jr., B.S.: PORs: Proofs of retrievability for large files. Proceedings of the 14th ACM Conference on Computer and Communications Security, 584–597 (2007).

35. Armknecht, F., Bohli, J.-M., Karame, G.O., Liu, Z., Reuter, C.A.: Outsourced proofs of retrievability. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 831–843 (2014).

36. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: INFOCOM (2010).

37. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security. 9, 1–30 (2006).

38. Pinkas, B., Reinman, T.: Oblivious RAM revisited. Advances in Cryptology – CRYPTO 2010. LNCS 6223, 502–519 (2010).

39. Bellare, M., Ristenpart, T., Rogaway, P., Stegers, T.: Format-preserving encryption. Selected Areas in Cryptography. LNCS 5867, 295–312 (2009).

40. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure deduplication. Advances in Cryptology – EUROCRYPT 2013. LNCS 7881, 296–312 (2013).

**Biographies**

James Alderman is a post-doctoral research assistant in the Information Security Group at Royal Holloway, University of London. His research interests primarily revolve around verifiable outsourced computation and the cryptographic enforcement of access control policies. He is currently employed on the European Union's Horizon 2020 project CLARUS aiming to develop a secure framework for storing and processing data outsourced to the cloud.

Jason Crampton is a professor of information security at Royal Holloway, University of London. His research focuses on access control, including models for access control systems, languages for specifying access control policies, and the cryptographic enforcement of access control policies. He served on the editorial board of ACM Transactions in Information and System Security from 2007 to 2012, and regularly serves on program committees for a wide range of information and computer security conferences.

Keith M. Martin is a professor in the Information Security Group at Royal Holloway, University of London. After research positions at the University of Adelaide and Katholieke Universiteit Leuven, he rejoined Royal Holloway in January 2000, became a Professor in Information Security in 2007 and Director of the Information Security Group between 2010 and 2015. Keith's current research interests include key management, cryptographic applications and securing lightweight networks. He is the author of “Everyday Cryptography” by Oxford University Press. As well as conventional teaching, Keith is a designer and module leader on Royal Holloway’s distance learning MSc Information Security programme, and regularly presents to industrial audiences and schools.

1. This chapter appears in Guide to Security Assurance for Cloud Computing and is published by Springer (http://link.springer.com/chapter/10.1007%2F978-3-319-25988-8\_2) [↑](#footnote-ref-1)