Crypto-integrity

Moti Yung

Department of Computer Science, Columbia University,
moti@cs.columbia.edu

Abstract. Designing cryptographic mechanisms and products is a challenging task. This task will become increasingly hard as software technology and systems evolve and as the new computational environment becomes more distributed, more diverse, and more global. In order to enable the inclusion of cryptographic components in the future infrastructure and within future applications, it is argued that assurance of their (secure) operation has to be provided and their robustness has to be exhibited in real time. This assurance, which we call *crypto-integrity* will guarantee the correct functioning of the cryptographic components in an efficient fashion. This built-in integrity should have no impact on the system security and should have minimal impact on its function, performance and composability.
We review the need for crypto-integrity in various known settings, ways to implement it based on known protocol techniques as well as potential future directions. The paper is written as a position paper and not as a survey of the vast relevant literature.

1 Introduction

Integrity assurance is a part of many modern cryptography constructions. In fact, cryptography itself is often employed to provide strong integrity such as "message integrity" (assured by hashing a message based on a secret key in a MAC operation). Other cryptographic operations have integrity associated with them, e.g. digital signing (initiated by Diffie-Hellman, Rivest Shamir and Adleman, and Rabin) involves a verification procedure which assures the authenticity of the signature.

The design of cryptographic protocols where many parties are involved in a joint activity allows dishonest adversaries to behave in arbitrary devious ways. Thus, the need to assure well behaved parties arises naturally. Early protocols like Rabin's signature protocol and Blum's coin flipping had assurance of behavior designed into them. Then, the development of the basic notion of zero-knowledge by Goldwasser, Micali and Rackoff was crucial to recognizing the central idea of systematic assurance of well behaved parties. The fact that NP languages have zero-knowledge proofs (and arguments) is fundamental and can be used to assure that actions taken in a cryptographic protocol are in accordance with the protocol specifications. This serves as a general plausibility result that integrity of parties in a protocol can be upheld and monitored.

Y. Zheng (Ed.): ASIACRYPT 2002, LNCS 2501, pp. 567–573, 2002.

However, for every protocol solving a specific task, we need to design specific proofs and integrity mechanisms that are efficient and suitable to its setting. In fact, as cryptographic services are deployed, every system configuration and every specialized setting will need to provide efficient and specialized methods for exhibiting the correct behavior of components; we call these methods *crypto-integrity*. Next we mention some systems factors affecting the need for specialized crypto-integrity.

1.1 System Setting and Requirements

We will now review some practical requirements and will argue how they can be achieved/ strengthened by having crypto-integrity functions.

As cryptographic primitives and protocols are being developed as products, their adoption into the computing and communication infrastructure will be based on their usefulness and effectiveness (typically measured in business by "Return on Investment"). This means that certain basic properties are required, some of which are related to generic desired properties (like user-friendliness), while others relate directly to the intrinsic properties of cryptographic design:

- Generality and composability of components: the basic product should be useful in many settings as a general primitive. We should be able to employ it with many (current and future) applications and it should remain secure in these settings.
- Adaptability (or scalable security): we should be able to embed the product in various settings (of different scales) and change its environment and even the threats to which it is susceptible, yet it should keep on being secure. (Many designs are too "setting specific" and are hard to adapt to new environments).
- Performance: this is an important factor that may fail the product when e.g. speed is a requirement or when it becomes too costly to implement fast or compact solutions. In some environments performance criteria are crucial (and this changes as technology changes).
- Assurance: there should be assurance about the product workings. Besides the proof of security which should be done in the proper setting of the entire application (end-to-end arguments), and besides testing, it will be useful if the product will have on-line assurance of the way its components work (namely, what we called crypto-integrity).

The above and similar requirements usually serve as a feedback to the crucial work on foundations of cryptography, where new notions are defined, designed and improved, and where the characteristic and inherent properties of the basic notions are investigated.

These requirements are also very useful to practitioners. To have a sustainable business one needs to have certain quality in its products. Having crypto-integrity may ease re-usability of components and shorten the test cycle. Having a general component that is adaptable to various settings and can support current and emerging applications is, at times, an important prerequisite for profit.

The notion of crypto-integrity has implications to all the above requirements. It helps assure the proper behavior of components, which means that parties are committed to certain computations, something that leads to predictable performance (and time-out mechanisms can further detect delays that are system specific methods to cope with delays caused by misbehaving components). It helps in exhibiting (at the interface between components) what is done by individual components and sub-systems, and thus helps in composing systems and the adaptability of components. It supports and reinforces formal assurance procedures such as certification of products by government bodies.

Crypto-integrity is an assurance mechanism which is achieved by enhancing the function of the component itself (in on-line operation). On-line integrity has many implications in special contexts. Let us mention one current implication. The context is a trusted computing environment run under a tamper resistant component of the architecture. This setting may have a lot of positive implications. It has however, many bad implications, if it is not run according to a "publicly agreed upon" specification. With crypto-integrity we may be able, at least partially, to assure compliance of a well specified trusted environment with its global specification (especially if we limit it to very specialized functions, since in general we cannot really tell what a tamper resistance cryptographic environment is doing as was shown by the notion of Kleptography [Young and Yung, Crypto 96, Eurocrypt 97, Crypto 97]).

In the rest of this paper I will mention examples of mechanisms (protocol design and settings) where crypto-integrity plays an important role.

2 Examples: the Usefulness of Crypto-integrity

2.1 Cryptographic Program Checking

Blum introduced the useful and elegant notion of *Program Result Checking*. In this setting, given arbitrary input α and program P, a checker C for a function f will catch, with high probability, if $P(\alpha) \neq f(\alpha)$. The checker has only "black-box" access to the program and accomplishes its goal on-line. Cryptographic program checking (developed in [Frankel, Gemmel and Yung, STOC96]) allows the on-line checking of programs computing cryptographic functions in a working environment.

In this model the checker worries about correctness (a concern that traditional "program checking" takes care of), since due to the adversarial setting we require correctness with very high probability. In addition, the owner of a program will output $P(\alpha)$ provided it is authorized to output the result, but the checker (user) learns nothing more about P from this checking procedure, in the spirit of the zero-knowledge complexity approach to knowledge. Such checking methods are witness-based (they allow the output of a few values to be known as a witness) and achieve fast verification. In some sense the procedures can be viewed as extending the "deniable signature" proof method of Chaum.

The basic application of this method is testing cryptographic servers. In the future, many servers will act on behalf of user populations and assurance of

non-spoofed service will be important. We now discuss several applications for cryptographic program checking.

Consider the *encrypting-machine requester game* where the encrypting-machine (server) is willing to encrypt authorized requests. If the checking process requires the encryption of other (unauthorized) plaintext so that the output of the request can be checked, then the checker can exploit this service to encrypt unauthorized texts as well. Cryptographic program checking can be used to prevent such an exposure.

Another similar application is the *international key escrow game*. This is related to specifications of key recovery methods between organizations and entities, an issue that is not yet well understood (on a technical level), but is similar to the concept of escrow encryption systems.

In this situation country A has a key escrow system and will allow country B to obtain decryption of messages of A's citizens under some predefined treaty and under some conditions. A does not want to provide B with the actual keys of its citizens (only the decrypted messages should be disclosed) while country B does not trust that A will reply with the correct cleartext values. Cryptographic program checking, in turn, allows B to verify the correctness of the outputs, while A knows that it is not being abused (by revealing messages not covered by the treaty or conditions). Of course, the setting is applicable to many (less controversial) scenarios. The basic ideas of our methodology can be applied to a *verifier hardware-device game*, where a holder of a result computed by some hardware device needs to probe a verifying device. For example, it makes it possible to make sure that a value computed in the past (a time stamp) is correct without the verification process leaking the computation itself (thus, recomputing the time stamp– which in effect causes an undesirable back-stamping). The methodology of cryptographic program checking applies to this situation as well.

2.2 Threshold Cryptosystems

One of the applications that motivated this research on cryptographic program checking is in the development of verification algorithms for threshold cryptography (where a function sharing or capability sharing is taking place). This is a method to distribute control of a function by a dealer or distributedly. In the *function sharing game* a function f is distributed amongst n agents as programs $P_1(\cdot), \ldots, P_n(\cdot)$ such that a threshold (quorum) of, say, any t are able to compute $f(\alpha)$ from $P_{i_1}(\alpha), \ldots, P_{i_t}(\alpha)$. There are several interesting applications for which function sharing is a very useful solution in practice (e.g., distributed decryption, signature generation, public key certification generation, e-cash generation, etc.).

Once the shares are available there is a polynomial-time combiner that collects the shares and combines them to the final result of the function. When agents misbehave this gives rise to the game between the agents and the combiner, where the combiner has to be sure to pick correct shares into its computation. If there is no efficient way to verify correctness of shares, the combiner may need to try all subsets of shares (but this will take exponential time). The

agent combiner game will assure the combiner which of the agents acted correctly. The need for *robust* function sharing was also expressed in an application for replicating services in a network were some of the clients and servers have been corrupted by an adversary. Since there is a real systems' need for the primitive, inefficient methods like non-interactive zero-knowledge techniques should be avoided. While many results have been achieved in this area, the applicability and usability of the results has yet to be realized.

2.3 Proactive Security

Another game, called the *proactive function sharing game*, is an extension of the robust function sharing game. In this system the agents' state is modified over time (by the agents themselves) so that an adversary may have access to all agents over the lifetime of the system but not to all (or not even to a quorum) at any particular point in time. The agents periodically modify their state so that information learned by a mobile adversary at two points in time is, for practical purposes, uncorrelated. In this game, the honest agents make sure that changing their state does not provide a means for the adversary to learn too much information nor to destroy the ability of honest agents to later compute the function. Hence each of the agents verifies that the information provided to it by other agents is correct before it changes states. This notion is called proactive security.

The notion of proactive security was a result of dealing with a mobile adversary which corrupts different parts of the system in different times. It was motivated by new threats like network viruses. It was first developed for the area of general secure multi-party computation (initiated by Yao and Goldreich, Micali and Wigderson) in [Ostrovsky and Yung, PODC 91]. It was somewhat motivated by an early notion of allowing users in this setting of general multi-party computation to leave and re-join the computation smoothly. (This last idea needed the method of "share-of-shares" which is a robustness mechanism was first employed in a work [Galil, Haber and Yung, Crypto 87]). It was also further motivated by Dijkstra's notion of self-stabilizing protocols which allows transient faults, whereas proactive protocols allows persisting faults (rather than transient) by introducing redundancy (requiring honest majority).

Many procedures have been "proactivized" and in particular so have many distributed cryptosystems. The need for integrity when we have the system dynamically changing and when honest users re-join, is crucial.

Proactive methods allow us to change the quorum of users that hold some computational capability distributedly within a system. This is a new function that is made possible by the built-in integrity mechanism which ensures the correctness of the shared capability, throughout.

2.4 Voting Schemes

Voting schemes have interesting requirements. They ask that the voter's action is universally verifiable yet his ballot has to remain secret. Various methods

assuring the integrity of the system and limiting malicious voters, preventing them from disturbing the global voting process have been developed. Recently (in [Kiayias and Yung, PKC 2002]) a small scale election with unique properties was given. It assures increased privacy (where in order to compromise the privacy of a ballot, all other voters have to collude against an individual) and combines it with a form of fault tolerance and universal verifiability in a way that there are no disputes in the on-line process (dispute freeness) due to the built-in crypto-integrity.

2.5 Assurance with respect to Off-line Third Parties

In escrow and key recovery systems (especially in auto recoverable cryptosystems, see [Young and Yung, Eurocrypt'98, PKC'00]), as well as in traceable e-cash and in various other settings, we have a user assuring that some action by an off-line third party is in fact doable, once this third party becomes active. The availability of public keys and the proof techniques which use them, enable such proof of actions by a third party where there is no need for on-line parties to participate. we expect such methods to find further applications in many areas.

2.6 Minimizing Key-Exposure

Recently, cryptosystems have been designed where key exposure is coped with either by "forwarding" (self updating) the key, making past keys inaccessible, or by sharing the key with a server (key-insulated cryptosystems developed in [Dodis, Katz, Xu and Yung, Eurocrypt'02]). We note that when sharing and updating keys with other elements, crypto-integrity is a must.

2.7 Multi User Setting

The case where multi users are present in a system (rather than two parties: a sender and a receiver) may change the setting and the possibility of procedures that can be employed. For example a "distributed proof" can be conducted assuring a group of users with honest majority of a fact while not revealing the underlying secret. Many areas of multi-user oriented cryptography are still open, and crypto-integrity is likely to play an increasingly important role in this setting.

2.8 Environmental Constraints and Protocols

Due to technological changes, the environment where protocols are being executed is changing. Protocol notions based on the Internet concurrency, and notions based on limited execution environments (mobile devices and smart cards) are being considered nowadays. These changes will affect the crypto-integrity requirements among other changes that they will dictate.

Environmental constraints also motivate research on modularity and composition of crypto protocols. The methods that assure integrity are paramount to

allowing properties like composition in specialized settings. For example, the issue of self-testing protocols while retaining the protocol's security is in its infancy (see [Franklin, Garay and Yung, DISC'99]).

3 Conclusions

We reviewed areas where crypto-integrity methods have been developed and used extensively. We showed how the notion allows for integrity assurance while retaining the secrecy of cryptographic techniques.

We claim that on-line assurance is a crucial security component: If we run tests of the systems in a working environment where we have to give up security to validate the system's correctness, we are at risk that someone in control of moving from test mode to operational mode can use this capability to compromise the system.

Also, on-line assurance is very important in working systems which evolve and change. It makes sure the core cryptographic component is acting correctly. Maintaining the integrity as the system changes is an interesting open area of research.

On-line crypto-integrity adds "function" by allowing parties to be off-line but nevertheless assuring that when they join they will be able to perform their task. With cryptography being part of many applications, such tools are expected to be crucial.

The research questions regarding on-line integrity in cryptographic settings are many: from improving the efficiency and other properties of existing methods, through questions related to new techniques and new primitives where integrity is crucial, to possibly new areas where crypto-integrity functionality is a must such as "safe cryptographic testing and development," "general notions of composability and modularity," "theory of reusable cryptographic methods," and "theory of update of system based on change of threats." We believe that given that "cryptosystems deployment as part of more general computing systems" is still (in spite of deployment successes) an area in its infancy, the area of assuring integrity in cryptographic settings is open to further investigation and to innovations.