Forward Security
Adaptive Cryptography: Time Evolution

Gene Itkis[⋆]

Computer Science Department
Boston University

Abstract. We survey the development of forward security and relate it to other concepts and trends in modern cryptography.
Ordinary digital signatures have an inherent weakness: if the secret key is leaked, then all signatures, even the ones generated before the leak, are no longer trustworthy. Forward-secure digital signatures were proposed to address this weakness: they ensure that past signatures remain secure even if the current secret key is leaked.
Similarly for the case of ordinary encryption, adversary that successfully exposed a secret key is typically able to expose even the old messages sent long before exposure. Forward-secure encryption ensures that the past messages are protected even if the current secret key is exposed.
We discuss forward security as a special case of key-evolving cryptography.

Keywords: cryptography, forward security, forward secrecy, forward-secure schemes, digital signatures, encryption, key exposures, public key infrastructures, PKI, revocation, recovery, key evolution.

[⋆] This material is based upon work supported by the National Science Foundation under Grant No. 0311485

Table of Contents

1 Security and Secret Keys

As our world is growing increasingly dependent on digital systems, security of these systems is becoming increasingly critical. In addition to accidental failures, threats of malicious attacks must be addressed by the security systems of today and tomorrow.

Connectivity of the digital systems has become an integral part of their functionality. However, connectivity could also provide malicious attackers with an easy access to the system, in particular allowing them to mount their attacks even from the other side of the globe.

Physical isolation is hardly ever an option in achieving protection, and so most systems must rely on other mechanisms for their security. These mechanisms, be they simple passwords authentication or sophisticated cryptographic tools, generally depend on maintaining some secrets (keys).[1]

Thus, security of a system hinges on the condition that the attackers cannot gain access to its secret keys. This condition may be difficult to satisfy, especially since these keys must be actively used by the system. One might try to make it harder for an adversary to expose the secret keys. To this end one might utilize special devices (such as smart-cards), multiple factor mechanisms (e.g., regular passwords, combined with smart-cards, and biometric mechanisms), etc. But our experience shows that no matter how strong is the protection of the secret keys, it is very likely that a sufficiently motivated adversary will succeed sooner or later and expose these keys. Thus, an experienced security systems designer will plan *explicitly* for the event of key exposures.

Therefore, the goals for a security system design can be formulated as threefold: (a) make key exposures as difficult and as expensive for adversaries as possible, (b) if/when the keys are exposed, minimize the damage; (c) recover from the exposures.

Modern cryptography offers a number of tools and techniques to assist in these tasks. These tools include threshold/proactive and forward-secure schemes. More recently, intrusion-resilient cryptography has been introduced, combining the proactive and forward-secure approaches. Key-evolution is one common theme in these techniques.

In this survey, we review these techniques, focusing on the forward-security.

The rest of the paper is structured as follows: Section 2 discusses digital signature and their limitation. Signatures schemes are then used throughout the survey as a default example for illustrations. Section 3 discusses various strategies for addressing security of the secret keys. Section 4 surveys the literature and history of the key concepts. The definitions for the key-evolving schemes are given in Section 5. And Sections 6, 7, and 8 survey various forward-secure schemes. Finally, Section 9 concludes with some open problems and other research directions related to forward-security.

2 Introduction by example: *Forward-Secure Signatures*

We use digital signatures as our standard example throughout this survey below. So, in the next subsection we introduce this important cryptographic tool.

2.1 Digital Signatures

Digital signatures are an important tool, critical to applications ranging from the widely used SSL/TLS to the more futuristic e-commerce, digital checks and digital cash. Digital signatures are discussed in greater detail in Sections 112 and 177 in this Handbook, as well as in much of the cryptographic literature.

[1] Auguste Kerckhoffs advocated that the security of the system be dependent only on the secrecy of these keys as early as 1883[79].

Intuitively, digital signatures are used to authenticate digital documents, much as the hand-written signatures are used to authenticate the documents written on paper. In the case of the traditional paper documents, the document contents and signature are bound together physically — by the paper that contains both. Even in the physical world the security of this binding can be questioned. But in the digital domain, such physical binding is simply non-existent. Thus, in the digital world the signature must be bound directly to the document content.

In both physical and digital case, there are two separate tasks: *signing* and *verifying*. The task of signing on behalf of a particular user should ideally be possible only for the user himself[2]. The verification typically should be possible for any member of the public.

The digital signature is thus characterized by the two algorithms: **Sign** and **Ver**. Both must take the document text (message) m as the input. The first, generates a signature σ, and the second takes that signature as the input.

It is convenient to use the same signing and verifying algorithms for all the users. Thus, both algorithms must also take as input some information that is unique to the particular user. For the case of the verification, this information should be public and is thus referred to as the user's *public key* $\mathtt{PK}$. Since only the particular user should be able to generate his signatures, the unique per-user information for signing must be secret and thus referred to as *secret* (or *private*) *key* $\mathtt{SK}$.

Some systems allow the $\mathtt{PK}$ to be simply the user's ID [117, 57, 56]. However, such identity-based schemes are often harder to construct, and therefore the association between the users and their public keys is often left out to the Public Key Infrastructures (PKI) (see, e.g., section 61 in this Handbook). In either case, the pair $\mathtt{PK}, \mathtt{SK}$ must be generated (perhaps from some additional parameters) – this is done by the algorithm **KeyGen**.

Thus, the following triplet gives the functional description of a signature scheme.

KeyGen$(1^k) \longrightarrow (\mathtt{SK}, \mathtt{PK})$**:** *key generation*
`Input:` a security parameter $k \in \mathsf{N}$ (given in unary)
`Output:` a pair $(\mathtt{SK}, \mathtt{PK})$, the secret key and public key

Sign$(\mathtt{SK}, m) \longrightarrow \sigma$**:** *signing*
`Input:` the secret key $\mathtt{SK}$ and the message m to be signed
`Output:` signature σ of m

Ver$(\mathtt{PK}, m, \sigma) \longrightarrow$ **valid|fail:** *verification*
`Input:` the public key $\mathtt{PK}$, a message m, and an alleged signature σ
`Output:` **valid** or **fail**. Usually, it is required that **Ver**$(\mathtt{PK}, m,$ **Sign**$(\mathtt{SK}, m)) \longrightarrow$ **valid**.

Intuitively, a signature scheme is secure if it is infeasible for an adversary without $\mathtt{SK}$ to compute any signature σ for a message m, such that **Ver**$(\mathtt{PK}, m, \sigma)$ outputs **valid**. This remains infeasible even if the adversary adaptively obtains legitimate signatures for many other messages of the adversary's choice. See [63] (or many subsequent papers and books) for precise definitions of signature security.

[2] It may be possible and desirable for the user to delegate authority vested in the signature to another party. In the physical world, this is usually achieved by explicit delegation of authority — e.g., power of attorney. Such delegations are also possible in the digital scenarios as well. But digital signatures potentially allow a stronger notion: a limited delegation of ability to generate signatures themselves. For example, using intrusion-resilient signatures [72] the user can enable another party to generate his signatures but only if they are tagged with a particular date.

2.2 Limitations of Signatures

Ordinary digital signatures have a fundamental limitation: if the secret key of a signer is compromised, all the signatures (past and future) of that signer become suspect. Even though the *signer* might know which signatures were issued by him and which by the impostor (using the stolen key), there is no way for the *verifier* to distinguish them.

Thus upon such a secret key compromise, the signer should revoke his public key (itself a non-trivial problem), and obtain a fresh key pair. But what to do with the already issued signatures (i.e., those issued before the compromise — in good faith)? Re-issuing them with the new key is expensive or even impossible (imagine having to do this for a certification authority, or in the absence of reliable and exhaustive records of the past signatures). What is even worse, a dishonest signer may see a key compromise as a golden opportunity to repudiate (some) previously signed documents. In fact, he might even fake a compromise himself (for example, by anonymously posting his secret key on the Internet and claiming to be the victim of a computer break-in).

3 Key Security

The above makes clear the importance of the security of the signer's secret key: both to prevent the actual key compromise as well as make the "faked" key compromises less believable.

Threshold Crypto: Space Dimension. One way to improve the security of the secret keys is to distribute it among different computers in such a way that breaking into individual computers would not affect security of the key, at least until sufficiently many of these computers have been compromised. This approach has been explored by *threshold* cryptography. For example, similarly to the secret-sharing scheme of Shamir [116], for any integers $n \geq t > 0$, the signing key $\mathtt{SK}$ can be shared among n computers, so that any t of them, working together, can generate valid signatures. But breaking into any $t-1$ of them gives *no* information about $\mathtt{SK}$ to the adversary (i.e., adversary learns *nothing* from these break-ins) ; in fact any $t-1$ shares of $\mathtt{SK}$ look completely random. The signing algorithm would then become a protocol executed by a sufficiently large subset (t or more) of these share-holding computers. The security definitions would have to be adjusted correspondingly — in particular, the $\mathtt{SK}$ cannot be reconstructed from the shares in any of these computers, since then it might be exposed (stolen) there.

Proactive Crypto: Time Evolution of Sharing. A particularly strong, *proactive* version of the threshold cryptography tolerates even compromises of all the computers — as long as they were not simultaneous. This is achieved by introducing an aspect of time evolution: while the threshold cryptography shared a secret key statically, the proactive cryptographic schemes would constantly re-share the keys. Thus the exposed key-shares would not help the adversary after that key sharing changed. To apply this to the above threshold signatures example, assume that at some point after an adversary breaks into a computer, the computer is completely "cleaned up", so the adversary looses control over that computer and access to the information on it (except the information, that the adversary already stolen). Then, the proactive techniques allow the signature scheme above to be strengthened to tolerate adversary eventually breaking into *all* n computers, as long as $< t$ are corrupted at a time.

Limits of Proactive Security. However, even in the case of proactive security, the "combined" state of the system remains unchanged — the secret key being shared is the same, only the specifics of how it is stored evolve with time. Therefore in the case of the total exposure (e.g. simultaneous compromise of all the computers holding key shares), the issue of past signatures remains.

Price of Proactive/Threshold Security. A major drawback of the threshold and proactive systems is that any transaction — such as generating a signature — requires at least t parties (i.e., participating computers) to perform joint computation: by executing a particular protocol to achieve the desired result. Such a communication can be inconvenient, inefficient, and sometimes even impossible. It also increases vulnerability of the participating computers.

3.1 Key-Evolution

Exploring Time Dimension. A radically different approach to protecting the keys is to securely erase them — a *securely erased* key cannot be stolen. So, the signatures generated with such a key cannot be repudiated by a fallacious claim of exposure (assuming that the key was not exposed prior to the erasure[3]). Then the signatures generated with the secret key before it was erased should remain trustworthy forever — even after the signer's computer is compromised. This approach motivates an essential evolution of the whole system to occur in time, where the past is fundamentally irreproducible.

Frequent Rekeying. The problem with erasing the secret key, of course, is that the signer can no longer produce signatures with it. With ordinary signatures, this means that the corresponding public key is now useful only for past signatures, and a new public key needs to be issued (and appropriately certified and disseminated) for the future ones. This makes such an approach expensive and inconvenient.

Forward-Secure Signatures. The goal of *forward-secure* signature schemes is to provide the benefits of frequent rekeying without incurring the costs of changing public keys (and associated overhead). They enable the signer to frequently erase the secret key while maintaining the same public key. The notion of a forward-security originated from the notion of "perfect forward secrecy" for key agreement [65], which protects past traffic even after long-term keys are compromised.[4]

To be more precise, in a forward-secure signature scheme the total time that the public key is valid (for example, one year) is divided into T *time periods* (for example, 365 days). At the end of each time period, the signer computes the next secret key from the current one (via a *key update* algorithm) and erases the current secret key. Each signature includes an essential new component: the time period during which the signature is issued. The forward-security property means that even if the adversary obtains the current secret key, she still cannot forge signatures for past time periods.

Limitations of Forward-Security. Just as for ordinary signature schemes, for the forward-secure signatures the signer is still expected to promptly *detect* key compromise. The sooner the determination is made and key revoked, the less damage the adversary can do. Thus, in particular,

[3] This assumption should not be taken lightly, of course. But it does offer a significant extra level of comfort. For example, imagine that a user decides to abbrogate a contract he signed a year earlier, by claiming that the secret key used to sign the contract has been stolen and the other party of the contract forged the signature using this stolen key. This claim would look much more dubious if the only way the alleged key compromise could have occurred is if it took place at least a year earlier. It is even more dubious if the signer has not discovered the key compromise until the terms of the contract began to look less attractive than they did at the time of the signing (and alleged key exposure!). While no full-proof security for the signatures appears to be possible, the secure erasure of the relevant key does seem to offer a greater degree of comfort. A slight further improvement of this comfort level is offered by other models, such as key-insulated and intrusion-resilient cryptography [46, 72], where the only time the key could have been stolen was on the day of the contract signing.

[4] The term "forward" in "forward-security" and "forward secrecy" is viewed by many as confusing. One can argue that "backward" would be more appropriate since such schemes protect transactions that happened *before* a key exposure. On the other hand, "forward" can be justified, because if a transaction occurs when the key is secure, then it will remain secure in the *future*, even after future key exposures.

forward-security does not remove the need for intrusion detection and for prompt key revocation (which both are rather problematic in practice).

Beyond Forward-Security. These limitations motivate research into extending the forward-security approach. Some extensions, such as intrusion-resilient and key-insulated schemes [72, 68, 44, 46, 47], aim to protect *future* as well as past signatures, thus making prompt detection of key compromise and key revocation less crucial for many applications. In particular, intrusion-resilient schemes allow to reduce the reliance on the revocation. Another extension, cryptographic tamper-evidence [69], addresses the issue of detection of key compromises. This extension capitalizes on the dynamically evolving nature of some of the schemes. Such evolution may enable detection of *use* of exposed keys even after a total compromise of the system (i.e., after *all* the secrets in the system have been exposed). Forward-security focuses on the evolution of the secret keys. However, there may be some benefits in evolution of the public keys as well — e.g., to allow a controlled repudiation of signatures generated with secret keys extorted from the user. This problem was considered in [103] and then further developed in [74], which not only restricted the possible repudiation to avoid its abuse, but also provided more efficient solutions. This was achieved by explicit generalization of the key-evolving schemes to include both secret key and public key evolution.

4 Threshold and Forward Security: Overview

In this section, we provide the survey of the bibliographic references to the main results in the areas of threshold and forward-security. We view these as the two dimensions along which the cryptography developed in response to the need to address key exposures.

4.1 Threshold and proactive cryptography

Much of the work focused on distributing secrets and computation in such a way that an adversary must commit multiple breaches before security of the system fails. Examples include the general and fascinatingly powerful (albeit inefficient in practice) multi-party computation methods (e.g., [59, 60, 18, 36]) and the more specific (and thus more efficient) *threshold* cryptography methods (e.g., [41, 108, 23, 55, 40, 127, 95, 118]).

Proactive security takes this approach further by forcing the adversary not only to break into many computers in order to learn anything, but also to accomplish all these breaks nearly simultaneously [107, 67, 40, 53, 32, 10, 33].

However, both threshold and proactive approaches require the cooperation of multiple parties for every cryptographic operation (even when there are some semi-trusted parties involved, as in [12]). It is difficult to apply them routinely in many settings, where the amount of data and processing demand greater efficiency and applications demand greater usability.

Another common approach to protecting secrets has been to keep them on computationally weak portable devices, such as smart cards. In order to make them usable for computationally intensive cryptographic tasks, approaches such as "server-aided" (e.g., [84, 109, 13, 104, 98, 75]) and "remotely-keyed" (e.g., [19, 20, 91, 21, 92, 125]) have been proposed (all-or-nothing transforms have been proposed as another solution to this problem, see [112, 26, 27, 120]). Furthermore, the devices can be made less likely to succumb to off-line attacks using the techniques of [94, 93].

The security against *partial* key exposures —when only some portion of a secret key gets into the adversary's hands— has been addressed in [31, 43, 48]. This approach may be useful in the cases when learning all the bits of a key is difficult for the adversary (e.g., for some devices, such as smart-cards, malicious probing of some bits might damage other bits — allowing the adversary to extract only partial information from the device).

None of the above approaches, however, addressed what happens in the case when the secret key is exposed: they simply increase the difficulty of stealing the secrets.

4.2 Forward-Secure Cryptography

Origins. Traditionally, forward-security (including the term itself) is traced back to the work of Christoph Günther [65] on key exchange protocols. This work, in particular proposes a notion of "perfect forward secrecy". This property essentially requires that the confidentiality of the past messages is not compromised even after the long-term secrets are exposed. This definition, however, does not in any way imply the key evolution, which we believe is fundamental to forward-security.

Continuing to focus on the forward secrecy (i.e., confidentiality of the past messages), Adam Back in cypherpunk email discussion proposed the idea of a public key cryptosystem, which would generate a sequence of private and corresponding public keys $\mathtt{SK}_i, \mathtt{PK}_i$ [9]. That is, ideally Back wished for two "non-reversible" functions f_1, f_2, such that $f_1(\mathtt{SK}_i) = \mathtt{SK}_{i+1}$ and $f_2(\mathtt{PK}_i) = \mathtt{PK}_{i+1}$. Of course, the messages encrypted for public key $\mathtt{PK}_i$ would still be decrypted with the private key $\mathtt{SK}_i$. But now, an adversary would not be able to decrypt those messages if she had only $\mathtt{SK}_{i+1}$. So, Bob could certify and advertise his public key $\mathtt{PK}_0$, and every day he could generate a new private key from the old one, securely erasing the old keys. Then Alice could send Bob a message on day i using key $\mathtt{PK}_i = f_2^i(\mathtt{PK}_0)$. Then even if Bob's key is exposed any time after day i, the message would remain confidential.

We can simplify the above notation to make it look more modern (i.e., matching the notation of [15]): without loss of generality, set $\mathtt{PK}_i \stackrel{\Delta}{=} \langle \mathtt{PK}, i \rangle$, where $\mathtt{PK}$ is *the* public key of Bob. Then the $\mathtt{PK}_i$ as defined by Back above, could be computed from $\langle \mathtt{PK}, i \rangle$ "on the fly" as $f_2^i(\mathtt{PK})$. This computation could be integrated into the decryption or signature verification algorithms as needed.

In the same draft, Back proposed a scheme which was not quite as good as the above ideal: while coming up with f_1 above was easy, f_2 turned out to be much harder[5]. So, instead of computing new public keys from the previous ones, Back's scheme computed them from the corresponding private keys. Then the sender of the message would simply be provided with all the public keys. In this case, f_1 could be any pseudo-random function, and the ElGamal/DH encryption [42, 50] would use the key $\mathtt{PK}_i = g^{\mathtt{SK}_i} \bmod p$, for some large prime p and a generator g of Z_p^*.

In the subsequent year, Ross Anderson proposed [7] to apply the same approach to signatures. He also observed that *identity-based* cryptosystem (those where the identity of a user served as his public key) can be adopted to provide forward-security: simply use combination of the public key and the date as the identity.

Key-Evolving Schemes and Forward-Secure Signatures. These initial ideas were formalized and further developed in by Bellare and Miner [15]. In particular, they provided the formal definitions for the key-evolving signature schemes and their security. The definitions we use below are closely base on the definitions of [15]. New forward-secure signature schemes were proposed in the same paper, followed by more constructions [83, 5, 71, 96, 82]).

Generic and Concrete Schemes. All the proposed schemes can be divided into two general categories: the generic and concrete.

[5] Indeed, only some seven years later this problem would be resolved, and even then only based on a less common bilinear Diffie-Hellman assumption [34]. Forward-secure public key encryption based on the standard DH, discrete log, or factoring assumptions remains an open problem to the day of this writing.

A generic scheme is constructed from any ordinary (non-forward-secure) scheme used as a black-box. The security of the forward-secure scheme is then reduced to the security of the underlying ordinary scheme (i.e., if some adversary can efficiently compromise the forward-secure scheme, then we can construct an adversary compromising the security of the underlying ordinary scheme). The efficiency of a generic scheme is usually measured in terms of the number of the invocations of the underlying ordinary scheme, the number of the ordinary keys, etc.

Both Back's and Anderson's schemes above, while originally presented as concrete schemes, could be generalized to generic ones.

A concrete scheme usually starts from a specific ordinary scheme and modifies it to achieve desired properties. A concrete scheme security is thus reducible to the specific cryptographic assumption, such as the strong RSA assumption, or hardness of discrete log, or bilinear Diffie-Hellman assumption, etc. The efficiency of the concrete schemes is measured in terms of the number of specific operations used (e.g., modular multiplications). As with many ordinary schemes, this necessitates expressing the costs in terms of two security parameters. For example, for the ordinary schemes such as Fiat-Shamir [52, 51], Guillou-Quisquater [64], and many others, one security parameter l characterizes the length of the modulus, while the other k determines the number of rounds, the length of the query, or the length of a hash function output. Typically, these differ by an order of magnitude: k is typically in the range 128–160, and l is usually in the 1024–2048 range.

Of the above schemes, [15] proposed both generic and concrete schemes; [9, 7, 83, 96] proposed generic constructions (though originally described as concrete); the constructions of [5, 71, 82, 34] are concrete.

Forward-Secure Public Key Encryption. Forward-secure public key encryption proved harder to achieve, and the first — and so far the only — result in that area was obtained only recently [34] (although some "approximations" of forward-secure encryption were put forward by [123, 46]).

Forward-Security for All Tasks. So far we really surveyed forward-security only for public key signatures and encryption. Providing forward-security for other tasks has also been considered.

For example, Krawczyk [83] constructed a generic forward-secure pseudo-random number generator, which he then used to construct a forward-secure public key signature scheme. Bellare and Yee [17] also provide definitions, careful analysis and constructions for forward-secure pseudo-random generators as well as for forward-secure symmetric signature and encryption schemes.

Forward-secure versions of group [119], threshold [4, 122, 90], and blind [49] signature schemes were defined and constructed (these are concrete constructions). One may also observe that the typical group key management schemes (also referred to as secure multicast) generally include forward-security in their definitions (see [126, 124, 35], and subsequent work); these however, are traditionally considered as a separate topic.

Also [3] generalized and analyzed the Fiat-Shamir paradigm of turning ID schemes into signatures. They extend their analysis to include forward-security.

5 Key-Evolution: Functional Definitions for Forward-Security

The main functional distinction of forward-secure versions of cryptographic tools is the explicit use of time in all algorithms (such as encrypting or signing). In addition, since the secret key in forward-secure schemes can evolve, there are procedures to handle such an evolution.

Below we outline the functional definitions for general key-evolving schemes that can include some mechanisms beyond forward-security. These definitions are based on [74], which in turn generalized the definitions of [15], where *key-evolving schemes* were introduced formally. We also separate here the key evolution aspect from the specifics of the cryptographic mechanism (e.g., key evolution is functionally independent from whether we deal with forward-secure signatures or forward-secure encryption).

5.1 Common Procedures

Key Generation. Key generation procedure generates typically a pair of keys $(\mathtt{PK}_0, \mathtt{SK}_0)$, for the initial time period 0. For symmetric cryptography schemes, the public key $\mathtt{PK}$ can be assumed to be null. This procedure is essentially similar to the corresponding one without evolution as in section 2.1: setting $T = 0$ makes the two completely identical.

KeyGen$(1^k, T) \longrightarrow (\mathtt{SK}_0, \mathtt{PK}_0)$**:** *key generation*

`Input:` a security parameter $k \in \mathsf{N}$ (given in unary), and the total number of periods, T,

`Output:` a pair $(\mathtt{SK}_0, \mathtt{PK}_0)$, the initial secret key and public key;

The maximum number of updates —the total number of time periods T— is required for some forward-secure schemes. Different approaches for eliminating this bound are considered in [96, 68].

Key Evolution. Key evolution procedure changes key $\mathtt{K}_t$ (which can be either $\mathtt{SK}$ or $\mathtt{PK}$) for the current period t into the key $\mathtt{K}_{t+\Delta}$ for the next time period $t + \Delta$ (typically, $\Delta = 1$, so the next time period is $t + 1$). This evolution can be either deterministic or randomized. This procedure is the one that is germane to the forward-secure (and other key-evolving) schemes.

KUp$(\mathtt{K}_t[, \mu][, \Delta]) \longrightarrow \mathtt{K}_{t+\Delta}[, \mu_t]$**:** *key update*

`Input:` the current key $\mathtt{K}_t$ [, and optionally, the update message μ] [, and, optionally, the number $\Delta > 0$ by which to increment the time period; by default $\Delta = 1$]

`Output:` The new key $\mathtt{K}_{t+\Delta}$ [and optionally, the update message μ_t];

Previously, only $\Delta = 1$ has been considered and unless stated otherwise, we will assume $\Delta = 1$ below. However, in some cases it might be beneficial to skip computing some of the keys. Naturally, for $\Delta > 1$ the update can be achieved by Δ successive key updates incrementing time by one. But in some cases more efficient updates are possible and desired.

The key evolution is typically done in a coordinated fashion: private and public keys evolve synchronously.

For the ordinary forward-security, the public key $\mathtt{PK}_t \overset{\Delta}{=} \langle \mathtt{PK}, t \rangle$ always consists of the invariant part $\mathtt{PK}$ and the current time period number t. Thus, the public key evolution is trivial: it simply increments the time period number. Evolution of the secret keys is usually more complex, for security reasons, and depends on the specific scheme.

For some schemes, such as in [74], it may be useful to allow greater freedom of the secret key evolution: e.g., make it randomized. This can increase the sophistication of the public key updates and require a greater level of coordination between the public and private key evolutions. The optional parameter/output μ in the **KUp** above provides the mechanism for such coordination: The private key update in this case generates an update message μ_t, which is then used as the optional input argument μ for the public key update.

Addition of time into the functional definitions of the standard cryptographic procedures is straight-forward. We include as an example such definitions for the case of the signatures.

5.2 Specific example: Forward-Secure Signatures

Functional Definition for Signing and Verifying. Finally, here we include the functional definition of the standard signing and verifying procedures for signatures, adapted for the forward security.

Sign($\mathtt{SK}_t, m$) $\longrightarrow \sigma_t$**:** *signing*

`Input:` the secret key $\mathtt{SK}_t$ for the current period t and the message m to be signed,

`Output:` signature σ_t of m (the time period t of the signature generation is included in σ_t);

Ver($\mathtt{PK}_t, m, \sigma$) $\longrightarrow$ **valid**|**fail:** *verification*

`Input:` the public key $\mathtt{PK}_t$, a message m, and an alleged signature σ (including the signing time period t'),

`Output:` **valid** or **fail**.

The verification procedure may impose additional conditions depending on the public key time period and the time of the signature generation, e.g., $t \geq t'$.

Thus, the full functional definition for the forward-secure signatures is obtained by combining the above **KeyGen**, **Sign**, **Ver** procedures with **KUp** applied to the $\mathtt{SK}_t$ (with no optional argument μ) at the end of each period t (no update message μ_t is generated in this case either). The public key update, which in this case simply increments the date, is usually not considered explicitly in the literature. But implicitly it is also assumed to be performed at the end of each time period (also without the optional input and output).

Note, that functional definition for ordinary (not forward-secure) signatures can be obtain from the above by eliminating all subscripts (denoting time) and omitting the key evolution **KUp**, leaving only the **KeyGen**, **Sign**, **Ver** procedures.

5.3 Security Definitions

The modern definitions of security tend to be based on adversary trying to distinguish certain inputs (e.g. encryptions of two known messages, or random vs. pseudo-random strings) or generate some output (e.g., a valid signature). A scheme is secure if no adversary could succeed with a probability significantly different than for a very simplistic adversary guessing at random.

The different types of security (e.g., known-plaintext, chosen-ciphertext, chosen-message, etc.) are captured by the specific powers of the adversary, including the information given to her. In this respect, forward-security introduces two issues into these security definitions: First, adversary queries are now to include references to the time period for which the query is made (e.g., for which an adversary-chosen message is to be signed in the chosen message attack).

Second, and more important, the adversary is given the secret key $\mathtt{SK}_t$ for the time period t of her choice. But then the adversary's attack, in order to be considered successful, is restricted to the time period $< t$.

This restriction is quite intuitive: Since the key-evolution algorithm is known, the adversary can use it to obtain secret keys $\mathtt{SK}_{t'>t}$ from the exposed $\mathtt{SK}_t$. Thus the restriction simply requires adversary to succeed in the period that is not trivially compromised. The details of which periods exactly are not "trivially compromised" might get more complicated in some other key-evolving models such as intrusion-resilient, or key-insulated.

Next, we consider different specific schemes. For each we sketch the security definitions and outline and/or survey the main constructions.

6 Forward-Secure Pseudo-Random Generators

6.1 Definitions

Notation. For any set S we write $x \overset{R}{\leftarrow} S$ to denote that x is chosen from the set S with the uniform probability (i.e., x chosen randomly from S). For strings a, b, let $a||b$ denote their concatenation, and $|a|$ the length of string a. Let adversary A be any probabilistic polynomial time (ppt) algorithm.

Pseudo-Random Generators (PRG) Definition. Intuitively, for parameters $k, m > k$, a function $G : \{0,1\}^k \longrightarrow \{0,1\}^m$ is a pseudo-random generator (PRG) if it is infeasible to distinguish its output from random. More precisely, set $s \overset{R}{\leftarrow} \{0,1\}^k$, $x_0 \leftarrow G(s)$, $x_1 \overset{R}{\leftarrow} \{0,1\}^m$, and $b \overset{R}{\leftarrow} \{0,1\}$. Then A's (distinguishing) advantage is defined as $|\mathsf{Prob}[\mathsf{A}(\mathsf{x_b}) = \mathsf{b}] - 1/2|$. G is a PRG for any ppt adversary A, if her distinguishing advantage is small[6] for sufficiently large k.

Forward-Secure PRG (fsPRG) Definition. To extend this to the forward-secure setting, we use the **KeyGen** and **KUp** procedures to generate and update the secret keys (seeds) $\mathtt{SK}_t$ for periods t. Without essential loss of generality and for the sake of avoiding extra notation, let the pseudo-random bits generated from $\mathtt{SK}_t$ in the period t be denoted as $\mathtt{PK}_t$.[7]

To define the advantage, for some period i, set $x_0 \leftarrow \mathtt{PK}_0||\ldots||\mathtt{PK}_{i-1}$, $x_1 \overset{R}{\leftarrow} \{0,1\}^{|x_0|}$, and $b \overset{R}{\leftarrow} \{0,1\}$. Similarly to the above, the advantage of A is $|\mathsf{Prob}[\mathsf{A}(\mathsf{x_b}, \mathsf{SK_i}) = \mathsf{b}] - 1/2|$ (note that A is also given $\mathtt{SK}_i$, so assuming deterministic key-evolution, A can generate all $\mathtt{SK}_{t>i}$ and thus $\mathtt{PK}_{t>i}$ on her own). And just as above, a key-evolving PRG is forward-secure if the advantage is small for all the ppt adversaries.

The period i above can be adaptively determined by the adversary, but choosing it at random would reduce the adversary's advantage at most by a factor of $1/T$.

6.2 Schemes

Many of the classical PRG schemes implicitly provide forward-security. However, this is not the case for some PRG schemes. For example, [17] shows that the alleged-RC4 PRG is not forward-secure.

Hard-core Iteration. A common approach to PRG constructions is to iterate a one-way permutation on the initial seed, at each iteration outputting a hard-core bit [22, 128, 58]. (Intuitively, a bit $b()$ is hard-core (or hidden) for a function $f()$ if $b(x)$ is not predictable from $f(x)$; see [22, 58] for full definitions.) The security proofs for such PRGs can usually be easily extended to include forward-security.

Indeed, let $\mathbf{KUp}(\mathtt{SK}_t)$ be a pseudo-random permutation, and let $\mathtt{PK}_t = b(\mathtt{SK}_t)$, where $b()$ is a hard-core bit for $\mathbf{KUp}()$. Then the hybrid proof of [128] implies the forward-security of the scheme.

Similarly, the proofs for the more general PRG of [66] based on any one-way function can be extended to include forward-security as well. In other words, their PRG is a fsPRG as well.

PRG Iteration. [83, 17] propose a construction of a fsPRG from any PRG. This construction is similar to the one above based on the one-way permutations, except two parts of the pseudo-random generator output is used in place of both the one-way permutation and hard-core bit (indeed, for a

[6] We leave the details of defining what exactly is to be considered small. Popular choices include exponential or super-polynomial.

[7] If the number of pseudo-random bits for each period is too large — e.g., if we wish to extend the definition to pseudo-random *functions* — then $\mathtt{PK}_t$ can be treated as the "daily seed", i.e., to be used as the seed for (another) pseudo-random generator/function (forward-secure or not) used to generate the bits for the period t. Of course, in this case $\mathtt{PK}_t$ must be sufficiently long and kept secret and erased at the end of the period t. This case is trivially reducible to the main case above.

random x, one-way permutation f with a hard-core bit predicate b, the string $f(x)||b(x)$ is pseudo-random). We present their construction in a slightly generalized form.

Let $G : \{0,1\}^k \longrightarrow \{0,1\}^{2k}$, and let G_0 denote the first k bits output by G, while G_1 — the last k bits (thus $G(s) = G_0(s)||G_1(s)$ for all s). For any bit b, define $F_b(s) \overset{\Delta}{=} G_b(s)$; and $F_{xb}(s) \overset{\Delta}{=} G_b(F_b(s))$.

Tree View. This can be viewed as in terms of trees: Consider a binary tree with each node having a label and a value. The root is labeled with an empty string; a left child of any node extends the parent's label with 0, while a right child extends it with 1. Let s be a value stored at the root, and let a node labeled l store $F_l(s)$. Clearly, the value of the left child of any node is computed by applying G_0 to the parent's value (respectively, G_1 for the right child).

Consider a completely unbalanced tree where each leaf has label 1^t0. This tree corresponds to the fsPRG construction of [83, 17]. Namely, $\mathtt{SK}_t = F_{1^{t+1}}(s)$ and $\mathtt{PK}_t = F_{1^t0}(s) = G_0(\mathtt{SK}_{t-1})$.

General Tree, Prefixless Construction. It is easy to generalize this construction, if desired, to other trees, including infinite ones. Specifically, consider any (possibly infinite) set of finite prefixless labels L: if $x, y \in L$ then x is not a prefix of y. Let l_i be the i-th smallest label in L. We can also refer to it as the prefixless (or self-delimiting) encoding of i according to L.

Now, we set $\mathtt{PK}_t \overset{\Delta}{=} F_{l_t}(s)$. The secret key $\mathtt{SK}_t$ must contain all the values from which the $\mathtt{PK}_{t'>t}$ can be derived.

For a node labeled l, define the *ancestor set* $P(l)$ to be the set of all the ancestors of l (excluding l). In other words, $P(l)$ are the nodes on the path from l to the root.

Define the *right set* $R(l)$ of l be the set of all the right children of $P(l)$ that are not in $P(l)$. $R(l)$ is the minimal set of nodes such that for any $l' \leq l$ $R(l)$ contains no ancestors of l', but for each $l'' > l$ it contains exactly one ancestor of l''. In other words, $R(l_t)$ is the minimum set of nodes from which the values for all $l_{t''>t}$, but for no $l_{t'\leq t}$ can be derived. Thus, as before, for each leaf l_t, we set $\mathtt{PK}_t \overset{\Delta}{=} F_{l_t}(s)$. Now, the $\mathtt{SK}_t \overset{\Delta}{=} \{\langle F_l(s), l\rangle | l \in R(l_t)\}$.

The above completely unbalanced tree is one example of prefixless encodings — essentially writing i in unary notation and marking its end with 0. Then for any leaf $l_t = 1^t0$, $R(l_t) = \{1^{t+1}\}$. Thus, that construction is a special case of our more general tree construction, using a very simple and inefficient prefixless encoding.

The efficiency of the encoding does not influence much the fsPRG constructions under consideration. More efficient schemes can be beneficial in the cases when many intermediate values need not be computed, that is when $\Delta > 1$ in the key update **KUp** (see sec. 5.1).

Moreover, essentially similar constructions are used later for other schemes (e.g., signatures) and will be able to benefit from better encodings.

Prefixless (Self-Delimiting) Encodings. Self-delimiting encodings are a well-known concept, playing an important role, in particular, in Kolmogorov Complexity [87–89], and in some scheduling problems, e.g., in [70]. In the context of forward-security they were utilized for the first time in [68] in a way similar to the one described here.

Various prefixless encodings abound. The simplest one handling infinite sets of inputs is probably to view each string t as an integer represented in unary (using only 1s), and delimited with a single 0: $l_t = 1^t0$. That's the one implicitly used above in the schemes of [83, 17] Of course, such an encoding is very inefficient: $|l_t| = 2^t$. However, a simple bootstrapping strategy can make it much more efficient: l_t can consist of the length $|t|$ of t, encoded in a prefixless notation, followed by t itself. In this case $|l_t| = 2|t| = 2\lg t$. Iterating this one more time we can get a prefixless encoding for which $|l_t| = \lg t + 2\lg\lg t = (1 + o(1))\lg t$. For practical purposes, it probably makes sense to stop here, though further iteration may yield better asymptotics (e.g. iterating $\lg^* t$ times, we can get $|l_t| = \lg t + \lg\lg t + ... + \lg^{\lg^* t} t + \lg^* t$).

Another approach to achieve prefixless encoding, (instead of using unary notation at the base step) can represent each bit as two bits, 0 as 00, 1 as 01, and use 11 to mark the end of the string. Clearly this encoding can be easily improved by encoding t in ternary and using 10 to represent the ternary digit 2. Then $|l_t| = 2\log_3 t < 1.6 \lg t$. Further reductions are possible: essentially, for any monotone function f such that $\sum 1/f \leq 1$ it is possible to achieve an encoding such that $l_t \leq f(t)$. In particular, for any $\varepsilon > 0$ there exists a prefixless encoding such that $|l_t| \leq (1+\varepsilon)\lg t + O(1)$. These methods can also be used in the same bootstrapping strategy.

6.3 Symmetric Cryptography: from fsPRG to fsMAC, Audit Logs, and Encryption

It is possible to define a forward-secure version of Message Authentication Codes (MAC) and symmetric encryption (see [17]). Intuitively the concepts are fairly straight-forward: the functional definition is obtained by adding the corresponding MAC or encrypt/decrypt functions to the common forward-security $\mathbf{KeyGen}, \mathbf{KUp}$ functions of the sec. 5.1.

The forward-secure constructions use the ordinary MAC or encrypt/decrypt functions and follow from the fsPRG directly: the key for each period is generated by the fsPRG and then used by the ordinary (non-forward-secure) functions. Namely, in the above notation, $\mathtt{PK}_t$ is used as an ordinary MAC or encryption/decryption key for the period t.

Another nice application for the forward-secure MACs —audit logs authentication— is suggested in [17] (a similar but more heuristic solution was also proposed in [113]). The basic idea is to authenticate a computer log using a forward-secure MAC. In this case, an intruder cannot modify the past entries into the log without being detected (having all entries numbered and equating the entry number with the time period eliminates undetected deletions as well). The log's verification, however, can be done on a separate machine which stores the original $\mathtt{SK}_0$ and can thus verify the log. In fact, forward-securely encrypting the logs may offer even better security.

Our more efficient prefixless construction can be beneficial in this application — many times an inspector may not be interested in accessing the log in a linear fashion. Rather it may be desired to extract specific log entries as fast as possible. In such cases, computing the desired $\mathtt{PK}_t$ from the root $\mathtt{SK}_0$ in the time logarithmic — rather than linear — in t would improve efficiency significantly.

6.4 Forward-Secure Public Key Crypto: Naive Generic Construction

Before we move on to discuss specific forward-secure public key cryptosystems, we discuss a simple but inefficient way of constructing a forward-secure cryptosystem (whether public key signatures, encryption or anything else) out of any ordinary one (same task, but no forward-security). Refer to the ordinary cryptosystem (and all its functions) used in the construction as the basic one(s). We shall use it as a black-box in our construction.

First, using the basic key-generation function, generate T public/secret key pairs $\langle p_t, s_t\rangle$ for all $t \in \{0, T-1\}$. Then publish all public keys p_t. For time period t use the ordinary scheme with the public/secret keys p_t and s_t. The forward-secure secret key $\mathtt{SK}_t$ for time period t is then $\mathtt{SK}_t = \{\langle i, s_i\rangle | i \geq t\}$. The forward-secure public key in general must contain all $\mathtt{PK}_t = \{\langle i, p_i\rangle | 0 \leq i < T\}$.

Such a naive scheme is very inefficient: it requires both public and secret storage to store T ordinary keys (for the secret storage this number is decreasing with time as $T-t$ keys stored at time t).

We can reduce the secret storage to one key plus whatever is required by the fsPRG: use fsPRG to provide the pseudo-random bits for the key-generation. Now, generate key pairs $\langle p_t, s_t\rangle$ twice:

after the first generation publish all the p_t. Then use the fsPRG re-generate the secret (and public) keys as needed for each period. This idea is further extended to obtain some of the generic signature constructions below.

7 Forward-Secure Signatures

Forward-secure signature schemes can be divided into two categories: those that use (in a black-box manner) arbitrary signature schemes, and those that modify specific signature scheme. We refer to the first category as *generic* or *black-box* constructions, and to the second as *concrete* or *explicit* constructions. The ordinary scheme(s) used as a black-box in the generic constructions is referred to as the *basic* scheme.

Below we consider these two groups separately.

7.1 Generic Constructions

In the generic constructions, typically a master public key is used to certify (perhaps via a chain of certificates) the basic public key used for each time period. Usually, this requires increase in storage space (compared to the basic scheme) by a noticeable —at least logarithmic— factor: to maintain the current (public) certificates and the (secret) keys for issuing future certificates.

Generic forward-secure schemes also require longer verification times than the ordinary signatures do: the verifier must verify the certification of the basic key for the current time period, in addition to verifying the actual signature with that basic key. The verification of the basic key can require multiple signature verifications — for each step of the certification chain. There is, in fact, a trade-off between storage space and verification time.

Static Schemes. The scheme proposed by Anderson [7] is the first scheme of this type. It uses a flat — one level — certification hierarchy. The public key PK for his scheme is the certification key. All the basic keys and their certificates are generated at the key-generation stage (certificate for each basic public key includes the time period for which this key is to be used). The certification secret key is then destroyed. The rest of the scheme is then pretty much the same as in sec. 6.4, except each signature is augmented by the certificate for the corresponding public key. Thus, for this scheme the public key is just a single basic public key. However the secret key must contain T basic secret keys and as many certificates.

Krawczyk [83] improves this construction, reducing the secret key storage to only the secret key of a fsPRG (in his construction a single secret). This is achieved by recomputing the secret keys with the help of the fsPRG as described above in sec. 6.4. This still leaves the storage of as many as T certificates.

Further optimization (also suggested in [83]) is to use the standard Merkle tree-hash construction [100]: Consider a finite binary tree; let each leaf store arbitrary value and each non-leaf store a hash of its children, for some cryptographic hash function. Then the root can serve as the certification key, where each leaf l is certified with the values in the nodes adjacent to its path to the root $P(l)$. This construction reduces both the secret and public key sizes to 1. However, its signature verification requires an additional T hash function evaluations, and an update at time t takes time linear in $T - t$.

The above two variants illustrate the trade-off between the storage space on the one hand and verification and update times on the other. However, using the recent results on traversing Merkle trees [121], it is possible to reduce all these cost — storage, verification and update — to logarithmic.

In all the above schemes, all the basic public keys are generated at the setup time — that's why we group them under the heading of static schemes. In particular, these schemes must have a known upper bound T on the number of time periods. Moreover, the setup time is proportional to T (for some of the above schemes, even the update time or storage are linear in T). Next we consider schemes which remove this limitation.

Dynamic Schemes. The above schemes constructed their certification tree completely at the setup time — next we consider schemes that construct the certification hierarchy on the "as needed" basis. That is, at each time the signer maintains the current basic key as well as (the minimal set of) the basic keys required to certify all the future (but not the past) basic keys.

The first such scheme was proposed by Bellare and Miner [15]. Essentially, they used a certification tree of height $\lg T$. This resulted in the verification needing to check additional $\lg T$ certification signatures. It was observed in [99] for standard (and in [5] for the forward-secure schemes, which use similar certificates) that each certifying public key is used only for authenticating a single message — the corresponding certificate of the children's public keys, and thus one-time signatures [85, 100] are sufficient for this purpose. These signatures offer greater efficiency (see, e.g., [111] and references there).

A few different certification topologies have been considered. A balanced $\lg T$ height tree used in [15] limits the total number of periods. This limit was extended to be exponential in a security parameter k by [96]. Their tree consists of a relatively small balanced tree each leaf of which is a root of another balanced tree. The height of a tree rooted at the i-th leaf of the top tree is i. Thus for a top tree of height $\lg k$ it contains k trees (for simplicity, assume k is a power of 2). Thus the whole tree contains $T = 2^k - 1$ leaves.

As pointed out in [68], both of the above tree structures represent special cases of prefixless encoding, and any other prefixless encoding will work for this purpose and may offer greater efficiency. In particular, using this approach, T is truly unbounded and the signatures at time period t require checking $\lg t + o(\lg t)$ certifications.

The same prefixless coding construction can be applied to the static schemes to reduce the signature length and verification time at time t to be logarithmic in t (as opposed to T in the original constructions). The cost of update is still likely to require time (and probably storage) logarithmic in T. This motivates a further study of applying the techniques of [121, 77] to traversing "slanted" Merkle trees (e.g., corresponding to the prefixless encodings).

While we can achieve compatible or even lower costs of running (but not setup) for the static schemes, the dynamic schemes still have advantage of being able to support unbounded number of periods (i.e., $T = \infty$), while the static schemes cannot handle even very large values of T, since the setup still takes $\Omega(T)$.

7.2 Concrete Constructions

A number of forward-secure schemes based on specific number-theoretic assumptions and ordinary signature schemes have been proposed. As most of the practical ordinary signature schemes, most of the proposed concrete forward-secure schemes are in the random oracle model. Clearly, any security proof in the random oracle model is no more than a "heuristic evidence" of security when the random oracle is instantiated by some cryptographic hash function, as is the common practice.[8] Forward-secure signature scheme, whose security is proven based only on a Strong RSA assumption (i.e., without the random oracle) was recently proposed by [30]. In addition to not relying on random

[8] Use of random oracles in cryptography is a topic of on-going debate. For example, [81] surveys some of the recent results intended to cast doubt on the random oracle model, but interprets them as actually supporting the model.

oracles, their construction allows a finer grain of forward-security. Namely, the signatures can be numbered within each period; then even the signatures issued during the exposure period but prior to the exposure remain secure. The other schemes in this section are proven secure in the random oracle model.

The forward-secure signature schemes based on factoring include those by Bellare and Miner [15] based on the Fiat-Shamir scheme [52]; and by Abdalla and Reyzin [5] based the 2^t-th root scheme [105, 106, 101]. The schemes by Itkis and Reyzin [71] based on the GQ signatures [64], and Kozlov and Reyzin [82] also require factoring to be hard, but it is not sufficient — Strong-RSA assumption is used (see [71, 82, 11, 54]). We are not aware of any forward-secure signature schemes based on discrete logarithm or DH assumptions.

The above concrete schemes are sufficiently sophisticated, that the in-depth coverage of these schemes is impractical in this survey. The reader is referred to the original articles for the details of the constructions and proofs. Here we restrict ourselves to the high-level discussion and comparison of the schemes.

Both [15] and [5] require signing and verification times that are linear in T.

The scheme of [71] reduces the signing and verifying times, signature sizes and storage costs to be essentially the same as for the ordinary GQ signatures, However, the update time of the Itkis-Reyzin scheme is proportional to T if the storage is kept minimum (one extra key). Increasing the storage to be logarithmic in T, the update costs can be reduced to logarithmic as well.

The Kozlov-Reyzin signature scheme improves the update time to just a single modular squaring, but at a high price: the verifier must now perform $2(T - t)$ multiplication per period (in addition to verifying each individual signature — the cost of that operation is compatible with the Itkis-Reyzin scheme, though is slightly greater). Thus this scheme can be efficient for the signer, esp. for the case of frequent updates (see [82] for the detailed performance comparison), but not for verifier (in fact, the verifier's cost exceed the signer's cost of [71], making it more compatible to (but better than) the costs of the Bellare-Miner and Abdalla-Reyzin.

Time Period Bound. All these schemes require at least some computation (at least in the setup stage) to be linear in T. Therefore none of the current non-generic constructions have the advantage of the dynamic generic schemes which allow unbounded number of time periods.

Time-Space trade-off: Pebbling. Both the Itkis-Reyzin and Kozlov-Reyzin scheme use an interesting technique of pebbling (suggested in [71]) to optimize certain computation costs at a modest (logarithmic) increase in storage cost. This technique is of independent interest, and in particular inspired (and is essentially equivalent to) the research into hash-chains computation [76, 37, 115].

8 Forward-Secure Public Key Encryption

In the section.6.4 above, we have outlined a trivial way of constructing various public key schemes (including encryption). That approach, however, is largely impractical, since it incurs linear in T costs even in the most fundamental parameters, most importantly, the public key.

The only scheme to date that overcomes such high costs is provided by Canetti, Halevi and Katz [34]. This scheme builds on the hierarchical identity-based encryption (HIBE) of Gentry and Silverberg [56], which in turn is based on Boneh and Franklin's identity-based encryption [24, 25]. All of these schemes are based on the Bilinear Diffie-Hellman assumption (BDH). As was the case for the concrete signature schemes, it is impractical to describe the construction and the proofs for the schemes, so we restrict ourselves to only the high level discussion and the reader is referred to the original articles for details.

Obtaining non-trivial public key encryption schemes based on more standard assumptions such as factoring, RSA, discrete log or Diffie-Hellman (DDH or CDH) are still an open problem.

Intuitively, HIBE schemes define a tree (possibly non-binary) such that the tree has a single "master" public key associated with the whole tree, and a secret key associated with each node. The encryption algorithm using the "master" public key and the id of any tree node v can encrypt a message so that it can be decrypted using the secret key for any of the ancestors of v. Moreover, the secret key for a node can be derived from the secret key of its parent. This derivation is one-way: the parent's secret key cannot be derived from a child's key.

The IBE and HIBE constructions of [24, 25, 56] needed to be extended slightly (which was achieved by some relaxation of security, which did not affect the forward-security scheme), resulting in a slightly different version of HIBE. The new HIBE scheme proposed in [34] had an additional advantage of being secure in the standard model (without the random oracles required in the previous constructions). Furthermore, the previous constructions were extended in [34] to (binary) trees of arbitrary depth.

With the above tools, the forward-secure encryption can use any of the prefixless encodings techniques much as in Sec. 7.1 above to obtain the costs proportional to $\lg t$ for any operation at time t (encryption, decryption or key update). The ciphertext and secret keys similarly grow linearly in $\lg t$.

9 Conclusion

9.1 Forward-Security: Open Problems

Forward-secure cryptography is a fairly well studied field, as might be seen even from this survey. However a number of interesting open questions remain.

Forward-Secure Public Key Encryption Based on RSA or DH. The forward-secure public key encryption schemes discussed in Sec. 8 is based on the Bilinear Diffie-Hellman (BDH) assumption, which is somewhat less "standard" than some of the other assumptions such as RSA, Strong-RSA, Decisional or Computation DH assumptions, etc. While groups where the BDH assumption appears to hold are known, these groups are not as simple as the Z_n^*. Therefore, it would be very interesting to find forward-secure encryption schemes that work in such groups and are based on the more common assumptions mentioned above.

Forward-Secure Signatures: Efficiency and Removing Random Oracles. Section 7.2 described some forward-secure signature schemes, whose signing and verifying is as efficient as for ordinary (GQ) signatures. However, some of the other operations costs for these schemes depended on T: while the memory and update costs were only logarithmic in T, the key generation was $\Theta(T)$. Can these overheads be reduced or even removed? Some generic schemes do achieve independence from T at the cost of introducing logarithmic (in current time period t) overhead in the signature size and the verification cost (there is no the overhead for signing, and the update overhead can be amortized to be just an additive constant). This may be acceptable in practice, but avoiding such overhead altogether would be much more appealing.

Most of the signature schemes deployed in real systems do not have a full security proof: their security proofs rely on using random oracles in the constructions. In real life, these oracles do not exist and are replaced with "cryptographic hash functions" or pseudo-random functions, such as MD5 or SHA1. This renders the security proofs inapplicable (see, e.g., [62], and the bibliography there). Signature schemes whose security proofs do not rely on random oracles have been proposed (e.g., the classic [63], or more recent and more efficient [38, 102]). Are there forward-secure variants of these schemes which have similar performance characteristics?

Static Generic Schemes: Traversing Slanted Merkle Trees. The recent algorithms for the efficient traversal of the Merkle trees [121] can improve performance of the static generic constructions for the forward-secure signatures. However, these still have the $\Theta(\lg T)$ overhead. The verification overhead of such static schemes can be improved to $\lg t$ for the current time period t by using the prefixless encodings constructions. Can the above Merkle tree traversal algorithms be adjusted for these prefixless trees? What is the best performance that can be obtained for such “slanted” trees traversals?

9.2 Evolving Cryptography: Beyond Forward-Security

As discussed in Section 3 forward-security provides an important tool for protecting the key and in a way represents a new fundamental development of cryptography: making it more resilient.

A number of papers (including those mentioned in sec. 4.2) have addressed adding forward-security to other schemes. For more examples of such combinations — solved and open — the reader is referred to [14]. While many of such combinations are very important, below we focus on what we see as building on a more fundamental aspect of forward-security.

We also discussed the time-evolution aspect of the forward-security. This trend had a number of subsequent developments.

Cryptographic Tamper-Evidence. No matter what security tools are utilized, a prudent security architect will always consider the case that the security is broken. In such cases, it is often desired to revoke the key, or undertake other similarly dramatic recovery steps (this might be required even for forward-secure systems, as discussed in sec. 3.1). Thus, it is important to detect when the security is broken.

Until recently, this tamper-detection was mainly achieved by heuristic methods, such as intrusion-detection systems. Indeed, it might seem that cryptography is powerless against an adversary who steals all the secrets. However, evolving cryptography opens some possibilities.

Any cryptographic system undergoes in important change after the adversary steals the secret keys of a user: the system would now have two players, where there is supposed to be one — the legitimate user, and the adversary with the stolen secrets, both playing the same role. These two players would actually “diverge” if at least one of them evolves in a randomized fashion. Furthermore, in some scenarios — such as signatures — both of these players might produce some output — e.g., legitimate and forged signatures). This makes it possible to detect the divergence.

More specifically, [69] defines *tamper-evident signature schemes* offering an additional procedure Div, which detects tampering: given two signatures, Div can determine whether one of them was generated by the forger (essentially, Div detects divergence of the legitimate user and the forger). Surprisingly, this is possible even after the adversary has inconspicuously learned some — or even *all* — of the secrets in the system. In this case, it might be impossible to tell which signature is generated by the legitimate signer and which by the forger, but at least the fact of the tampering will be made evident.

[69] defines several variants of tamper-evidence, differing in the power to detect tampering. In all of these, an equally powerful adversary is assumed: she *adaptively* controls *all* the inputs to the legitimate signer (i.e., all messages to be signed and their timing), and observes *all* his outputs (i.e., all signatures); she can also adaptively expose *all the secrets* at arbitrary times.

[69] provides tamper-evident schemes for all the variants and proves their optimality by showing tight lower-bounds. These lower bounds are perhaps more surprising than the constructions. The

lower-bounds proofs are information-theoretic and thus cannot be broken by any methods (including by introducing any number-theoretic or algebraic complexity assumptions).

General Key-Evolution: Recovery. As we mentioned above, a seasoned security architect will always address the case of security compromise. The first issue to address — detecting the compromise — is discussed above. But then recovery mechanisms from such compromises are required. For digital signatures such a recovery should ideally —and when possible— include invalidation of the signatures issued with the compromised keys.

Recovery from one special case of compromises — extortion – were considered in [103]. There, so called *monotone signatures* were defined. In monotone signatures, a user can be forced (e.g., at a gun point) to release the secret key. In this case, the user releases a "fake" secret key. This key generates signatures which look perfectly valid under the current public key. However, at a later time (once out of the danger) the user updates his public key. Under the new public key, all the signatures generated by the user before or after the extortion remain valid. But the signatures generated by the adversary using the extorted key become invalid under the new public key.

[74] generalizes this work. It considers two models for key exposures: full and partial reveal. In the first, a key exposure reveals *all* the secrets currently existing in the system. This model is suitable for the pessimistic inconspicuous exposures scenario. The partial reveal model permits the signer to conceal some information under exposure: e.g., under coercive exposures the signer is able to reveal a "fake" secret key. The monotone signatures assume the partial reveal model in this terminology.

[74] propose a definition of *generalized key-evolving signature scheme*, which unifies forward-security and security against the coercive and inconspicuous key exposures (previously considered separately [15, 103, 69]). We based our definitions in this review in part on the definitions of [74].

The new models helped to address certain repudiation problems inherent in the monotone signatures [103], and achieve performance improvements.

Intrusion-Resilience: Space-Time Combination. Forward-secure cryptography protects the past time periods even when the current secret key is exposed. But what about protecting the future? Key-insulated cryptography [46, 47] attempted to provide some security to the future. In that model the user is partitioned into two modules: *actor* and *base*. This is somewhat akin to the threshold cryptography model. However, unlike the threshold model, all the transactions are performed by the actor alone. Only the updates require the base to communicate with the actor, sending it an update message. Without such a message, the actor cannot update itself to the next period — thus an adversary who steals actor's secret key but does not receive the update message loses her advantage at the end of the time period, as the stolen secret key expires. Exposures of the base alone do not provide any useful information to the adversary.

However if the adversary exposes both base and actor at arbitrary times (and in some variants, even with sufficiently many exposures of only the actor), all the security is lost, including the past.

Intrusion-resilient model [72] eliminated these security limitations. This model also uses the two modules: actor and base. However it really combines all the benefits of the forward-security and pro-active crypto (see sec. 4.1). Thus, the intrusion-resilient model can truly be considered as a time-space combination.

In particular, intrusion-resilient model tolerates arbitrarily many break-ins into both actor and base in arbitrary order. As long as these break-ins are not simultaneous, the only time periods that are compromised are those for which the actor was exposed[9]. Moreover, if a simultaneous

[9] The concept of exposure must be extended to include the trivial indirect exposure: When the actor is exposed at time period t and adversary intercepts all the messages from the base, including the next update message, then clearly, the actor should be considered exposed at time $t + 1$ as well.

exposure does occur, then the past periods remain secure (the future in this case obviously cannot be protected, since the adversary now knows the full state of the system).

The granularity of the exposures is determined by the frequency of *refresh* messages, similar to those used in the pro-active security. Namely, if there is at least one refresh message not seen by adversary between two exposures, then these exposures are not considered to be simultaneous. If the update message must be sent to from the base to the actor exactly once in a time period, the refresh messages can be sent as frequently as desired — more when the user feels his system is under the attack.

Intrusion-resilient schemes for signatures [72, 68], and for public-key encryption [44, 45] have been proposed. [72] builds on the concrete construction of [71], while [68] proposes a generic scheme based on an arbitrary ordinary signature scheme. Similarly, for the public key encryption, [44] proposes a scheme based on specific algebraic assumptions, while [45] presents a generic construction based on any *forward-secure* encryption scheme with certain homomorphic properties.

Interrelations between intrusion-resilient, key-insulated, forward-secure and so called proxy signatures were recently studied in [97].

9.3 From Theory to Practice

Key-evolving cryptographic schemes offer potential benefits in a number of practical areas. Applications to Public Key Infrastructures (PKI) for intrusion-resilient and key-insulated signatures were suggested in [73, 86], respectively. Other applications could be timed commitments, bidding, time-stamped delegation, and many others.

Many of these applications can have a significant practical impact. This is especially true for the case of PKI related applications, since PKI is already widely deployed and used.

This transfer from theory to practice could be facilitated in particular by extending various cryptographic standards to accommodate key-evolving cryptography. Additionally, integrating the key-evolving cryptographic tools into the popular protocols (e.g., such as SSL/TLS) would also increase security and potentially even offer infrastructure savings.

In fact, most of the cryptographic tools deployed today already include (implicitly and/or "out-of-band") some form of key evolution in the form of key revocation, periodic changes of the secret keys (e.g., as in pay-TV data streams) or even password changes. Explicit integration of key evolution promises to leverage existing key-evolving crypto tools, as well as enable and stimulate future productive research and progress in this direction.

Glossary

Authentication: Typically, assurance that a message was sent (authorized) by the purported author (and was not modified in transmission). Authentication mechanisms include *public key* (*digital signatures*) and *symmetric* (*MACs*) cryptography tools.

CA: See *Certification authority.*

Certificate: a document associating a particular *public key* with a particular entity (user, corporation, privilege, etc.). Certificate is typically a record containing a *public key* and the entity it is associated with, *signed* by a *Certification Authority.* The entity (or its representative) typically holds the secret *private key* corresponding to the certified public key.

Certification authority (CA): An authority, trusted within the appropriate *PKI* to issue and manage *certificates.*

Certificate revocation: A mechanism to revoke certificates: identify those certificates which should not be trusted (e.g., if the corresponding private key has been *exposed*).

Cryptographic hash: An efficiently computable function h, for which it is infeasible to find $x \neq y$ such that $h(x) = h(y)$. Cryptographic hash must be a *one-way function*, but a one-way function is not necessarily a cryptographic hash.

Hash tree: A tree where the leaves are associated with some input values, and for each internal node, its value is the cryptographic hash of its children.

Key exposure: An event of an adversary learning the secret (private) key.

MAC: See *Message Authentication Code.*

Message Authentication Code (MAC): A symmetric cryptography mechanism for message *authentication.* Typically implemented as a keyed *one-way function* or *cryptographic hash.*

Non-repudiation: A cryptographic service that (legally) prevents the originator of a message from denying authorship/authorization at a later date. This service can be generalized to other transactions.

Merkle tree: See *hash tree.*

One-way function: An efficiently computable function, but whose inverse is not efficiently computable.

PKI: See *public key infrastructure.*

Private key: A value kept secret by its owner and used to perform operations that are intended to be available only to him/her: e.g., digitally sign his/her messages and/or documents, or read (decrypt) secret messages intended only for this private key owner. A private key is typically associated with a *public key.*

PRG *or* **PRNG** : See *pseudo-random number generator.*

Pseudo-random: Indistinguishable from random.

Pseudo-random number generator (PRNG): An algorithm computing a pseudo-random string from a random (but shorter) *seed.*

Pseudo-random function: A function of a seed and an index, such that for a random seed the string of the function outputs for all the indices is pseudo-random.

Public key: A publicly available value associated with a particular secret *private key*, and used to perform operations that are intended for general public: e.g., verifying a digital signature, or encrypting a message for a particular user (the holder of the corresponding private key). A public key is often associated with the owner of the corresponding *private key*, e.g., by means of a *certificate* within a particular *PKI.*

Public key cryptography: a collection of algorithms and protocols utilizing related pairs of *public* and *private keys*, used for complementary operations: e.g., signing with the private key, while verifying the signature using the corresponding public key; or encrypting using public key and decrypting using the corresponding private key. Computing outputs of private key operations should be infeasible without the proper private key.

Public key infrastructure (PKI): A mechanism for associating public keys with entities such as users, corporations, privileges, authorizations, etc. PKI typically includes servers, logistical mechanisms and policies, and implies particular trust relations. PKI typically consists of a set of *certification authorities (CAs)* administering *certificates*: their issuing, maintaining, and revoking.

Random function: A function whose outputs are random: unpredictable for any input value, even when given the outputs for all the other input values.

Random oracle: A random function given as a "black box". Random oracle is an abstract construct, in practice often emulated using a cryptographic hash. Proofs of security of some cryptographic mechanisms assume access to a random oracle; such proofs of security may not apply when the random oracle is implemented as an actual algorithm. Security of such mechanisms is a subject of much debate and research.

Repudiation: An attempt to disavow a prior commitment, authorization, or transaction: e.g., an attempt to deny *signing* a particular document.

Signature: A value used to authenticate a message as authorized by the *signer.* The *signer* computes the signature as a function of the message being signed and the signer's *secret (private) key.* The signature can be *verified* using the *public key* associated with the signer's private key. Computing a valid signature (the one which verifies as valid with a proper public key) should be infeasible without the private key.

Symmetric cryptography: A collection of algorithms and protocols utilizing a single secret key, such that complementary operations, such as encrypting and decrypting, use the same key. This is in contrast to *public key cryptography* where analogous complementary operations require different keys.

References

1. *Fourth ACM Conference on Computer and Communication Security.* ACM, Apr. 1–4 1997.
2. *Progress in Cryptology — CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science.* Springer-Verlag, 2003.
3. M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT ' 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433, Amsterdam, The Netherlands, 2002. Springer-Verlag, Berlin Germany.

4. M. Abdalla, S. Miner, and C. Namprempre. Forward-secure threshold signature schemes. In D. Naccache, editor, *Progress in Cryptology — CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer-Verlag, Apr. 8-12 2001.
5. M. Abdalla and L. Reyzin. A new forward-secure digital signature scheme. In T. Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129, Kyoto, Japan, 3–7 Dec. 2000. Springer-Verlag. Full version available from the Cryptology ePrint Archive, record 2000/002, `http://eprint.iacr.org/`.
6. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.
7. R. Anderson. Invited lecture. In *Fourth ACM Conference on Computer and Communication Security* [1]. (see also [8]).
8. R. Anderson. Two remarks on public key cryptology. Technical Report UCAM-CL-TR-549, University of Cambridge, Computer Laboratory, Dec. 2002. http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-549.pdf.
9. A. Back. Non-interactive forward secrecy, 1996. Posting to cypherpunks mailing list (6/9/1996), archived at http://cypherpunks.venona.com/date/1996/09/msg00561.html.
10. B. Barak, A. Herzberg, D. Naor, and E. Shai. The proactive security toolkit and applications. In G. Tsudik, editor, *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 18–27, Singapore, Nov. 1999. ACM Press.
11. N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 11–15 May 1997.
12. D. Beaver. Server-assisted cryptography. In *New Security Paradigms Workshop (NSPW '98)*, pages 92–106, New York, Sept. 1999. Association for Computing Machinery.
13. P. Beguin and J.-J. Quisquater. Fast server-aided RSA signatures secure against active attacks. *Lecture Notes in Computer Science*, 963, 1995.
14. M. Bellare. An automatic crypto research topic or paper title generator. http://www.cs.ucsd.edu/users/mihir/crypto-topic-generator.html.
15. M. Bellare and S. Miner. A forward-secure digital signature scheme. In M. Wiener, editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer-Verlag, 15–19 Aug. 1999. Revised version is available from `http://www.cs.ucsd.edu/~mihir/`.
16. M. Bellare and B. Yee. Forward security in private key cryptography. Report 2001/035, Cryptology ePrint Archive, 2001. http://eprint.iacr.org/2001/035.ps.gz.
17. M. Bellare and B. Yee. Forward security in private key cryptography. In *CTRSA: CT-RSA, The Cryptographers' Track at RSA Conference, LNCS*, 2003. Originally appeared as [16].
18. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [6], pages 1–10.
19. M. Blaze. High-bandwidth encryption with low-bandwidth smartcards. Technical report, AT&T Bell Laboratories, Oct. 1995. ftp://research.att.com/dist/mab/card_cipher.ps. Draft.
20. M. Blaze. High-bandwidth encryption with low-bandwidth smartcards. In D. Grollman, editor, *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, pages 33–40, Cambridge, UK, 21–23 Feb. 1996. Springer-Verlag.
21. M. Blaze, J. Feigenbaum, and M. Naor. A formal treatment of remotely keyed encryption. In *Advances in Cryptology – EUROCRYPT '98*, pages 251–265, 1998.
22. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–863, Nov. 1984.
23. D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In *Proc. 17th International Advances in Cryptology Conference – CRYPTO '97*, pages 425–439, 1997.
24. D. Boneh and M. Franklin. Identity-based encryption from the Weil Pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO ' 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.
25. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, June 2003.
26. V. Boyko. On the security properties of OAEP as an all-or-nothing transform. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 503–518, 1999.
27. V. Boyko. *On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, May 2000. http://theory.lcs.mit.edu/ cis/theses/victor-phd.ps.gz.
28. G. Brassard, editor. *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990, 20–24 Aug. 1989.

29. C. Cachin and J. Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques Interlaken, Switzerland, May 2-6, 2004 Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.
30. J. Camenisch and M. Koprowski. Fine-grained forward-secure signature schemes without random oracles. In *International Workshop on Coding and Cryptography*. INRIA and ENSTA, 2003.
31. R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In Preneel [110], pages 453–469.
32. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 98–115, 1999.
33. R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, 13(1):61–105, Jan. 2000.
34. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In E. Biham, editor, *Advances in Cryptology—Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*. Springer, 2003. Also available at Cryptology ePrint Archive, Report 2003/083, `http://eprint.iacr.org/2003/083`.
35. G. Caronni, M. Waldvogel, D. Sun, N. Weiler, and B. Plattner. The VersaKey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, Sept. 1999.
36. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [6], pages 11–19.
37. D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '02)*, volume 2357 of *Lecture Notes in Computer Science*, Hamilton, Bermuda, 2002. International Financial Cryptography Association (IFCA), Springer-Verlag, Berlin Germany.
38. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.
39. I. B. Damgård, editor. *Advances in Cryptology—EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991, 21–24 May 1990.
40. Y. Desmedt. Some recent research aspects of threshold cryptography. In *Proc. 1st International Information Security Workshop*, pages 158–173, 1997.
41. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In Brassard [28], pages 307–315.
42. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
43. Y. Dodis. *Exposure-Resilient Cryptography*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Aug. 2000. http://www.toc.lcs.mit.edu/ yevgen/ps/phd-thesis.ps.
44. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung. Intrusion-resilient public-key encryption. In *Progress in Cryptology — CT-RSA 2003* [2], pages 19–32.
45. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung. A generic construction for intrusion-resilient public-key encryption. In *Progress in Cryptology — CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 81–98. Springer-Verlag, 2004.
46. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In Knudsen [80].
47. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC'03)*, 2003.
48. Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In B. Pfitzmann, editor, *Advances in Cryptology—EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 301–324. Springer-Verlag, 6–10 May 2001.
49. Duc, Cheon, and Kim. A forward-secure blind signature scheme based on the strong RSA assumption. In *ICIS: International Conference on Information and Communications Security (ICIS), LNCS*, 2003.
50. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985, 19–22 Aug. 1984.
51. U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
52. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 Aug. 1986.
53. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Proc. 17th International Advances in Cryptology Conference – CRYPTO '97*, pages 440–454, 1997.
54. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. S. Kaliski Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 17–21 Aug. 1997.
55. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 12–16 May 1996.

56. Gentry and Silverberg. Hierarchical ID-based cryptography. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 2002.
57. M. Girault. An identity-based identification scheme based on discrete logarithms modulo a composite number. In Damgård [39], pages 481–486.
58. O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.
59. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 174–187, Toronto, Ontario, Canada, 27–29 Oct. 1986. IEEE.
60. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.
61. S. Goldwasser, editor. *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 21–25 Aug. 1988.
62. S. Goldwasser and Y. T. Kalai. On the (in)security of the Fiat-Shamir paradigm. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 2003.
63. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
64. L. C. Guillou and J.-J. Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Goldwasser [61], pages 216–231.
65. C. Günther. An identity-based key-exchange protocol. In Brassard [28].
66. J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
67. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Fourth ACM Conference on Computer and Communication Security* [1], pages 100–110.
68. G. Itkis. Intrusion-resilient signatures: Generic constructions, or defeating strong adversary with minimal assumptions. In SCN02 [114].
69. G. Itkis. Cryptographic tamper evidence. In *10th ACM Conference on Computer and Communication Security*. ACM, Oct. 27–30 2003. Also avaliable from `http://www.cs.bu.edu/~itkis/papers/`.
70. G. Itkis and L. A. Levin. Power of fast VLSI models is insensitive to wires' thinness. In *30th Annual Symposium on Foundations of Computer Science*, pages 402–407, Research Triangle Park, North Carolina, 30 Oct.–1 Nov. 1989. IEEE.
71. G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In J. Kilian, editor, *Advances in Cryptology—CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354. Springer-Verlag, 19–23 Aug. 2001.
72. G. Itkis and L. Reyzin. SiBIR: Intrusion-resilient signatures, or towards obsoletion of certificate revocation. In Yung [129]. Available from `http://eprint.iacr.org/2002/054/`.
73. G. Itkis and L. Reyzin. SiBIR: Intrusion-resilient signatures, or towards obsoletion of certificate revocation. In Yung [129]. Available from `http://eprint.iacr.org/2002/054/`.
74. G. Itkis and P. Xie. Generalized key-evolving signatures, or how to foil an armed adversary. In *1st MiAn International Conference on Applied Cryptography and Network Security*. Springer-Verlag, 2003.
75. Jakobsson and Wetzel. Secure server-aided signature generation. In *PKC: International Workshop on Practice and Theory in Public Key Cryptography*. LNCS, 2001.
76. M. Jakobsson. Fractal hash sequence representation and traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02)*, pages 437–444, July 2002.
77. M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. Fractal Merkle tree representation and traversal. In *Progress in Cryptology — CT-RSA 2003* [2].
78. C. kang Chu, L. shan Liu, and W. guey Tzeng. A threshold GQ signature scheme. Jan. 28 2003. See `http://eprint.iacr.org/2003/016`.
79. A. Kerckhoffs (von Nieuwenhof). La cryptographie militaire. (French) [Military cryptography]. *Journal des Sciences Militaires*, Jan. 1883. Available at `http://www.petitcolas.net/fabien/kerckhoffs/`.
80. L. Knudsen, editor. *Advances in Cryptology—EUROCRYPT 2002*, Lecture Notes in Computer Science. Springer-Verlag, 28 April–2 May 2002.
81. N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. `http://eprint.iacr.org/`.
82. A. Kozlov and L. Reyzin. Forward-secure signatures with fast key update. In SCN02 [114].
83. H. Krawczyk. Simple forward-secure signatures from any signature scheme. In *Seventh ACM Conference on Computer and Communication Security*. ACM, Nov. 1–4 2000.

84. C.-S. Laih, S.-M. Yen, and L. Harn. Two efficient server-aided secret computation protocols based on additional sequence. In *Proc. Advances in Cryptology Conference – AISACRYPT '91*, pages 450–459, 1991.
85. L. Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, October 1979.
86. Z. Le, Y. Ouyang, J. Ford, and F. Makedon. A hierarchical key-insulated signature scheme in the ca trust model. In *Information Security (ISC 2004)*, volume 3225 of *Lecture Notes in Computer Science*, page 280.
87. L. A. Levin. On the concept of a random sequence, in Russian). *Doklady Akademii Nauk SSSR (Proceedings of National Academy of Science of USSR)*, 5(14):1413–1416, 1973.
88. L. A. Levin. Laws of information conservation (non-growth) and aspects of the foundations of probability theory, in Russian). *Problemy Peredachi Informatsii*, 3(10):206–210, 1974.
89. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 1993.
90. Liu, Chu, and Tzeng. A threshold GQ signature scheme. In *International Conference on Applied Cryptography and Network Security (ACNS), LNCS*, volume 1, 2003. See also [78].
91. S. Lucks. On the security of remotely keyed encryption. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop*, volume 1267 of *Lecture Notes in Computer Science*, pages 219–229, Haifa, Israel, 20–22 Jan. 1997. Springer-Verlag.
92. S. Lucks. Accelerated remotely keyed encryption. In L. Knudsen, editor, *Fast software encryption: 6th International Workshop, FSE'99, Rome, Italy, March 24–26, 1999: proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 112–123, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1999. Springer-Verlag.
93. P. MacKenzie and M. Reiter. Networked cryptographic devices resilient to capture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 12–25, Oakland, CA, May 2001. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.
94. P. MacKenzie and M. K. Reiter. Delegation of cryptographic servers for capture-resilient devices. Technical Report 2001-37, DIMACS, Nov. 1 2001. ftp://dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/2001/2001-37.ps.gz.
95. M. Malkin, T. Wu, and D. Boneh. Experimenting with shared generation of RSA keys. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '99)*, San Diego, CA, Feb. 1999. Internet Society.
96. T. Malkin, D. Micciancio, and S. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Knudsen [80].
97. T. Malkin, S. Obana, and M. Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In Cachin and Camenisch [29], pages 306–322.
98. J. Merkle. Multi-round passive attacks on server-aided RSA protocols. In S. Jajodia and P. Samarati, editors, *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-00)*, pages 102–107, N.Y., Nov. 1–4 2000. ACM Press.
99. R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer-Verlag, 1988, 16–20 Aug. 1987.
100. R. C. Merkle. A certified digital signature. In Brassard [28], pages 218–238.
101. S. Micali. A secure and efficient digital signature algorithm. Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, Cambridge, MA, March 1994.
102. D. Naccache, D. Pointcheval, and J. Stern. Twin signatures: an alternative to the hash-and-sign paradigm. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 20–27, Philadelphia, PA, USA, Nov. 2001. ACM Press.
103. D. Naccache, D. Pointcheval, and C. Tymen. Monotone signatures. In P. Syverson, editor, *Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 2001.
104. P. Nguyen and J. Stern. The Beguin-Quisquater server-aided RSA protocol from Crypto'95 is not secure. *Lecture Notes in Computer Science*, 1514, 1998.
105. K. Ohta and T. Okamoto. A modification of the Fiat-Shamir scheme. In Goldwasser [61], pages 232–243.
106. H. Ong and C. P. Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In Damgård [39], pages 432–440.
107. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *10-th Annual ACM Symp. on Principles of Distributed Computing*, pages 51–59, 1991.
108. T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 8–11 Apr. 1991.
109. B. Pfitzmann and M. Waidner. Attacks on protocols for server-aided RSA computation. *Lecture Notes in Computer Science*, 658, 1993.

110. B. Preneel, editor. *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*. Springer-Verlag, 14–18 May 2000.
111. L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with optimal signing and verifying. In *ACISP '02: 7th Australasian Conference on Information Security and Privacy*, July 2002.
112. R. Rivest. All-or-nothing encryption and the package transform. In E. Biham, editor, *Fast software encryption: 4th International Workshop, FSE '97, Haifa, Israel, January 20–22, 1997: proceedings*, volume 1267 of *Lecture Notes in Computer Science*, page ??, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1997. Springer-Verlag. Also on http://theory.lcs.mit.edu/ rivest/fusion.ps.
113. B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, 1999.
114. *Third Conference on Security in Communication Networks (SCN'02)*, volume 2576 of *Lecture Notes in Computer Science*. Springer-Verlag, Sept. 12–13 2002.
115. Y. Sella. On the computation-storage trade-offs of hash chain traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '03)*, Lecture Notes in Computer Science, Hamilton, Bermuda, 2003. International Financial Cryptography Association (IFCA), Springer-Verlag, Berlin Germany.
116. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
117. A. Shamir. Identity-based cryptosystem and signature scheme. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO ' 84*, volume 196 of *Lecture Notes in Computer Science*, pages 120–126. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1985.
118. V. Shoup. Practical threshold signatures. In Preneel [110], pages 207–220.
119. D. X. Song. Practical forward secure group signature schemes. In *Eighth ACM Conference on Computer and Communication Security*, pages 225–234. ACM, Nov. 5–8 2001.
120. D. R. Stinson. Something about all or nothing (transforms). *Designs, Codes, and Cryptography*, 22(2):133–138, Mar. 2001.
121. M. Szydlo. Merkle tree traversal in log space and time. In Cachin and Camenisch [29], pages 541 – 554.
122. Tzeng and Tzeng. Robust forward-secure signature schemes with proactive security. In *PKC: International Workshop on Practice and Theory in Public Key Cryptography*. LNCS, 2001.
123. W.-G. Tzeng and Z.-J. Tzeng. Robust key-evolving public key encryption schemes. Report 2001/009, Cryptology ePrint Archive, 2001. http://eprint.iacr.org/2001/009.ps.gz.
124. D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures. Internet Request for Comment RFC 2627, Internet Engineering Task Force, June 1999.
125. R. Weis, B. Bakker, and S. Lucks. Security on your hand: Secure filesystems with a "non-cryptographic" JAVA-ring. *Lecture Notes in Computer Science*, 2041, 2001.
126. C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 68–79, 1998. Appeared in ACM SIGCOMM Computer Communication Review, Vol. 28, No. 4 (Oct. 1998).
127. T. Wu, M. Malkin, and D. Boneh. Building intrusion-tolerant applications. In *Proceedings of the 8th USENIX Security Symposium (SECURITY-99)*, pages 79–92, Berkely, CA, Aug. 23–26 1999. Usenix Association.
128. A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 Nov. 1982. IEEE.
129. M. Yung, editor. *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science. Springer-Verlag, 18–22 Aug. 2002.