

On the Performance, Feasibility, and Use of Forward Secure Signatures

Cronin, Jamin, Malkin & McDaniel
or Cronin et al



Abstract

- **Contributions:**
 - first, a new methodology for comparing the performance of signature schemes, and
 - second, a thorough examination of the practical performance of FSS. We show that for many cases the best FSS scheme has essentially identical performance to traditional schemes, and even in the worst case is only 2-4 times slower.
- On the other hand, we also show that if the wrong FSS configuration is used, the performance can be orders of magnitude slower.
- **Categories and Subject Descriptors**
 - E.3 [Data Encryption]: Public key cryptosystems
- **General Terms**
 - Performance, Design, Security
- **Keywords**
 - forward-secure signatures, digital signatures



Key Exposure Problem

- Signature-based systems are very vulnerable to the *key exposure problem*, which in practice is a far more likely cause of compromise than cryptanalysis.
- Once a private key has been exposed, not only are all future signatures associated with the compromised key suspect, but all *past* signatures as well, since there is no secure way to tell that a signature was generated before or after the compromise.
- All previous signatures must thus be indirectly revoked.
- The damage of these compromises can be enormous, both in terms of overhead in revoking and reissuing all past signatures, and in terms of the new security vulnerabilities introduced by the possibility of repudiation.



Fraud & Certificate Authority Compromise

- **The compromise of a Certificate Authority's (CA's) root private key**
 - results in *all* certificates from that CA being unverifiable until clients are updated with the new root public key, not just new certificates signed by the replacement key.
 - All existing certificates (signed public keys) must also be revoked and re-certified with the new key, as there is no way to verify that they were signed prior to the compromise.
 - Where the root CA is popular (e.g., Verisign), the compromise could lead to widespread disruption of the Internet.
- **The digital signing of legal contracts illustrates another environment where the possibility of key exposure severely weakens security.**
 - By purposely exposing their private key, a party that wishes to back out of a contract gains the ability to repudiate their previous signature at any time by claiming it to be forged.



Electronic Checks

- As another example, consider an electronic checkbook application, in which each check is created by the account holder signing an amount, a recipient, and the current date using the checkbook's private key.
- The recipient submits the electronic check and the signature to the account holder's bank, and the check is honored if the signature is deemed valid.
- Here, if the private key is exposed, it is incumbent upon the bank and account holder to investigate and possibly repudiate every check received by the bank, possibly even valid uncashed checks.



Metrics

- **One of the challenges of performing this type of study is the lack of systematic metrics for evaluating the inherent tradeoffs of signature systems.**
- **Instead of simply looking at individual operation costs in our study, as one of the contributions of this work we develop an evaluation metric in which operation costs are amortized based on expected use over the lifetime of the key(s).**
- **We believe this new usage-based metric is more robust because it encompasses the application and environment, not just cycle consumption of basic operations.**



Simple Solutions

- Consider the following trivial forward-secure signature scheme, with a parameter T denoting the total number of time periods over which the scheme is supposed to operate.
- Starting from any standard signature scheme as a base, the signer runs the key generation protocol T times, obtaining T secret/public key pairs $(sk_1, pk_1), \dots, (sk_T, pk_T)$.
- The public key is now set to $PK = (pk_1, \dots, pk_T)$, while the secret key (for the first time period) consists of $SK_1 = (sk_1, \dots, sk_T)$.
- For each time period $1 \leq j \leq T$, signing and verifying are performed using the base scheme relative to the secret key sk_j (for signing) and public key pk_j (for verifying).
- To update from time period j to $j+1$, the signer simply erases the key sk_j (which is no longer necessary).
- **This trivial scheme is clearly not practical, because it requires public and secret key sizes which are linear in T , the number of time periods the schemes can be used for.**



Outline

- An outline for the remainder of the paper follows.
- In the next section we overview of FSS schemes and their theoretical (asymptotic) performance characteristics.
- We then present our empirical study in the following section, exploring the actual performance of these schemes.
- Section 4 describes the reference implementation of FSS used in our study, as well as several unexpected issues which arose during its development.
- Finally, we conclude and summarize our findings and present ideas on future work.



Efficiency Requirements

- What are the efficiency requirements when designing a forward secure signature scheme?
 - We cannot hope to beat standard signature schemes, since forward-secure signature schemes are a strictly stronger construct.
 - Thus, we would like a forward-secure signature scheme to perform not much worse than a standard one, in terms of time (for key generation, signing and verifying), and space (key size and signature size).
- In particular, it is not desirable for these parameters to grow with the number of **time periods T**.
- One might be tempted to require that there is no dependency whatsoever on T. But signature schemes already depend on some security parameters, which must be super-logarithmic in T.
 - The “**super-logarithm**,” $\lg^*(n) = \min i \text{ such that } \exp^*(i) \geq n$.
 - Remark: Note that \exp^* grows very rapidly, whereas \lg^* grows very slowly: $\exp^*(5) = 265536$, while $\lg^*(265536) = 5$.
 - Thus $\lg^*(n) \leq 5$ for all “practical” n .
 - However, eventually $\lg^*(n)$ goes to infinity as n does, but at an almost unimaginably slow rate of growth.
- If this were not the case, by the time period T is reached, the scheme could have been broken by exhaustive search.



Bellare-Miner Tree

- **Bellare and Miner [5] suggest a forward-secure scheme based on a binary certification tree.**
 - Roughly speaking, a binary tree is constructed with T leaves, where each leaf corresponds to a time period.
 - An instance (public-key, secret-key pair) of a base (standard) signature scheme is associated with each node in the tree.
- **The public key of the forward-secure scheme is the public key of the root, and the secret key consists of all the base key pairs associated with nodes on the path from the root to the current time period.**
- **A signature for time period j consists of a certification chain from the appropriate leaf to root, where the leaf key is used to sign the actual message, and each node is used to sign the public key of its child.**
- **Verification proceeds by verifying each signature on the chain, up to the top level (root) signature which is verified against the public key.**



Bellare-Miner Tree (cont.)

- In this scheme, all performance parameters (size and time) are $\text{poly}(k, l)O(\log T)$, where k, l are the security parameters (the actual dependence on k, l depends on the performance of the underlying base scheme).
- As explained above, this logarithmic dependence on T is very good, and, at least from a theoretical point of view, this scheme is satisfactory.
- Still, this is not competitive with standard signature schemes, as signing a message involves $\log T$ signatures (which are expensive public-key operations) in the underlying base scheme.
- Allowing a high number of possible time periods T , this factor can be prohibitive.
- On the other hand, key update is very efficient, roughly consisting of two (amortized) key generations in the underlying base scheme (since each key in the tree is only generated once).



Product Construction

- Malkin, Micciancio, and Miner suggest two composition operations, taking any two forward-secure schemes with T_1 , T_2 time periods, respectively, and constructing a new forward-secure signature scheme with more time periods.
- These constructions are suggested as tools in constructing flexible forward-secure scheme by applying them repeatedly in different combinations, possibly with other known schemes.
- The constructions are also used toward the main scheme.
- Their first composition operation is the product construction, resulting in a scheme with $T_1 \cdot T_2$ periods.
 - Here, an instance of the first scheme (with T_1 periods) is generated, and for each time period, a new instance of the second scheme (with T_2 periods) is generated underneath it.
 - The public key is the public key of the top (first) scheme, and a signature consists of signing the actual message with the key of the second scheme (in the appropriate time period), and signing the public key of the second scheme with the key of the first scheme.



Product Construction (cont.)

- The product construction can be viewed as making explicit the main building block that was already used by the Bellare-Miner tree (as well as Anderson's original scheme, and other certification based constructions).
- Indeed, when iterated recursively, starting from any standard base scheme, the product construction results in a scheme which is essentially the same as the Bellare-Miner tree.



Sum and Iterated Sum Constructions

- The second composition operation suggested is the sum construction, combining two schemes with T_1 , T_2 time periods, respectively, to a scheme with T_1+T_2 time periods.
 - Here, an instance of each of the two schemes is generated, and the public key is the hash of both public keys.
 - A signature consists of the two public keys, and a signature of the message according to the first (if it is within the first T_1 time periods), or the second (if it is within the next T_2 periods).
- Taking the secret key to consist of the (secret and public) keys of both schemes would allow the right functionality for the signer, but result in an inefficient key size.
- Instead, a more efficient way to generate and maintain the secret keys is proposed, and the reader is referred to [28] for details.
- When iterated recursively, starting from any standard base scheme, the iterated sum construction results in a binary tree, with each time period associated to a leaf, similar to the Bellare-Miner tree.



Sum and Iterated Sum Constructions (2)

- Each node in the tree is the hash of its two children, rather than being used to sign its children.
 - This means that signing (and verifying) is much more efficient, as $\log T$ hashes (private key operations) are extremely fast to compute, compared with $\log T$ signatures, which are slow.
 - On the other hand, key update is much slower here, since the signer must generate all the public keys in a bottom-up fashion, in order to compute their hash which is the public key; since the signer does not keep all keys (or else the key size would be very large), regeneration of keys is necessary during the life time of the scheme, resulting in (amortized) $\log T$ key generations per update.
 - In contrast, in the Bellare-Miner scheme, the tree is built top down, with new keys generated only when needed, resulting in 2 key generations per update.
- The asymptotic performance of the iterated sum construction is also $\text{poly}(k, l) \log T$; when explored more carefully, signing and verifying is extremely efficient, consisting of one signature in the base scheme plus $\log T$ hashes, while key generation and updates take about $\log T$ key generations of the base scheme, which may be slow ([28] suggests that this may be improved using onetime signatures[14, 8]).



MMM Tree

- The main scheme is to use an iterated sum construction with a polynomial number of time periods, where for each time period, another iterated sum construction is attached, using the product construction.
- The number of periods for each iterated sum construction on the lower level keeps increasing as time progresses, starting from a 2-period scheme attached to the 1st time period of the top level, then a 4-period scheme attached to the 2nd period, an 8-period scheme for the 3rd period, and so on.
- The MMM construction achieves a new feature that previous schemes did not, namely that the maximal number of time periods, T , need not be fixed in advance, and thus does not influence the performance.
 - More time periods are added as needed, for an arbitrarily large polynomial number of time periods.
 - The performance slowly degrades, proportional to $\log t$, where t is the number of time periods elapsed so far.
 - In terms of performance, signing consists of two signatures plus $\log t$ hashes in the underlying base scheme (which is very efficient), while key update consists of $\log T$ key generations in the base scheme (which may be expensive).



MMM Tree (cont.)

- The main performance advantage of MMM, namely that of efficient signing and verifying, is due to the fact that the iterated sum construction is prominently used.
- The use of product construction may in principle make MMM a little less efficient than iterated sum, but it allows for unbounded number of time periods, which also makes the performance better, as it only depends on the number of time periods elapsed so far, and not on the maximum T . This is what the theoretical analysis of the schemes indicates.
- An experimental performance analysis is conducted in this paper, and compares MMM, iterated sum, product, and Bellare-Miner tree in different settings.



Specific versus Generic Schemes

- The main advantage of the generic schemes is that they have provable security (assuming any signature schemes exist).
- This is in contrast to the specific schemes which can only be proven secure in the random-oracle model, a weakness they inherit from the underlying standard signatures they are based on.
- Proven security in the random-oracle model is very valuable as a heuristic for security, but provides a significantly weaker security guarantee.
 - In particular, it does not even guarantee the existence of an instantiation of the random oracle for which the scheme is secure.
- Another advantage of generic schemes stems from the fact that they can be used with any underlying base signature schemes.
 - This provides a stronger security guarantee (as it requires a weaker assumption – any implementation of one-way functions suffices).
- Furthermore, this allows for flexibility in optimizing and trading off the time and space parameters of the FSS scheme, by using base schemes with different performance characteristics, rather than being bound to the properties of a specific base scheme.



Performance Evaluation

- **Digital signature performance, whether it be for traditional or forward-secure signatures, is all about tradeoffs.**
 - **Balancing these tradeoffs in order to select an optimal set of parameters is not always a straightforward task.**
 - **In this section, we present several metrics for formalizing these tradeoffs between different aspects of signature performance for both traditional and forward-secure signature schemes.**
- **To our knowledge, no previous work has utilized such a technique capable of evaluating signature schemes' performance over the entire spectrum of uses.**
- **We then evaluate the performance of signature schemes with regard to these metrics using benchmarks of several of the FSS schemes introduced in the previous section.**



Performance Metrics

- **Key generation (or update) time, signature time, and verification time are all indicators of a signature scheme's performance.**
 - However, no one aspect alone is enough to judge whether one signature scheme is better than another for all situations.
- **Many earlier performance comparisons take an informal approach at resolving this problem by first looking at a specific situation and then picking which operation seems to be most important for it.**
 - This works well for simple situations but does not help when it is unclear which operation is most important or in seeing the entire picture with regards to performance tradeoffs.
- **Instead of taking a similar approach for our analysis, we look at what characteristics make a given environment using signatures unique, and how to express this as a set of parameters.**
- **We define several metrics using these parameters which compute a single amortized cost for the performance, allowing us to make direct comparisons between schemes for any given situation.**
- **Using this evaluation framework, we are able to not only look at specific cases as previous performance studies have done, but also at how the performance changes over the entire range of parameters, and how different signature schemes perform in this broader picture.**



Traditional Signature Schemes

- Key Generation: G , Cost: C_g
- Signature: S , Cost: C_s
- Verification Rates: V , Cost: C_v
- Ratio1 (R_1) = Verification / Signature
- Metric 1 = $C_v + 1/R_1 + C_s$
- Ratio2 (R_2) = Signature / Key Generation
- Metric 2 = $C_s + 1/R_2 * C_g$
- Metric 3 = $C_v + 1/R_1 * C_s + 1/R_1R_2 * C_g$



Forward Secure Signature Schemes

- New operation key update U with cost C_u to replace key generation parameter G .
- No change for Metric 1 ($M1$)
- Modified $R2$, $M2$, $R3$ and $M3$ are denoted with a star*: $R2^*$, $M2^*$, $R3^*$ and $M3^*$
- $R2^* = \text{Signature / Update}$
- $M2^* = C_s + 1/R2^* * C_u$
- $M3^* = C_v + 1/R_1^* C_s + 1/R_1 R_2^* C_u$



Experimental Setup

- **Four generic FSS**
 - Bellare-Miner (BMTree)
 - Iterated Sum (ISum)
 - Product (Prod)
 - MMM
- **Three base Algorithms**
 - RSA
 - DSA (Digital Signature Algorithm)
 - Elliptic Curve DSA (ECDSA)
- **1.5 GHz Pentium 4 w/1 gig RAM running FreeBSD 4.8**



Base Signature Scheme Performance

Short-Term Security			
	RSA	DSA	ECDSA
	1024	1024	t163k1
keygen (C_g)	352	5,500	3.68
sign (C_s)	10.5	4.36	3.75
verify (C_v)	0.540	5.35	7.61
Long-Term Security			
	RSA	DSA	ECDSA
	1536	1536	t223k1
keygen (C_g)	862	58,500	6.72
sign (C_s)	27.6	8.78	6.81
verify (C_v)	1.03	11.1	13.6

Table 1: Base signature scheme performance, in milliseconds.



Forward Secure Signature Scheme Performance

Short-Term Security								
	BMTree 256 RSA	ISum 256 RSA	Prod 16*16 RSA	MMM 255 RSA	BMTree 256 ECDSA	ISum 256 ECDSA	Prod 16*16 ECDSA	MMM 255 ECDSA
keygen	5,630	83,300	10,500	2,980	123	949	124	40.8
sign (\mathcal{U}_s)	10.5	10.4	10.5	10.5	3.75	3.74	3.73	3.79
verify (\mathcal{U}_v)	4.47	0.576	1.08	1.08	66.7	7.45	14.9	14.9
update	635	1,280	959	1,310	14.4	14.8	11.5	15.1
gen+up (\mathcal{U}_u)	654	1,610	997	1,320	14.9	18.5	12.0	15.2

Table 2: Forward-secure signature scheme performance (first 255 periods), in milliseconds.

- Short term refers to key size believed safe against brute force attacks today.



Unknown Maximum Period

- The results for Cs are not very interesting, as the cost of signing is completely dominated in all four FSS schemes by the single base signature performed.
 - Regardless of how many periods used or what the maximum period of the key is, the cost of signing is the same as the cost for the base algorithm signature.
- For Iterated Sum, Product and MMM, the same holds true for the cost of verification.
 - Similar to signing, all three of these schemes require a fixed number of verifications regardless of the current period or the maximum period.
- Bellare-Miner Tree, however, is dependent on the maximum period (but not the current period) for the number of base verifications it must perform, and therefore Cv is dependent as well.
- Table 3 shows the average verification cost for the ten Bellare-Miner Tree configurations tested.

Max Periods	RSA	RSA	ECDSA	ECDSA
256	4.47ms	8.60ms	66.7ms	123.4ms
512	5.00ms	9.53ms	74.0ms	137.1ms
4096	6.35ms	12.4ms	94.7ms	176.1ms



Implementation Challenges

- **Deterministic Key and Signature Generation**
 - Necessary for FSS
 - OpenSSL and AES-128 PRNG
- **Secure Deletion of Sensitive Key Material**
 - Encrypt before write
 - Secure management of many private keys
- **Protecting Against Timing Attacks**
 - side-channel attacks
 - choose specific inputs and measuring the time between request and reply



Conclusions

- In this paper we have explored the practical performance characteristics of forward-secure signature schemes.
 - We define a new framework for comparing signature schemes which takes into account the application environment in computing an amortized cost for basic operations.
 - We use this tool to compare several different FSS schemes built using generic constructions, as well as several non-forward-secure signature schemes used as bases for these constructions.
 - We use our performance metrics to examine a number of example uses for forward-secure signatures, and provide recommendations as to the optimal FSS scheme and configuration to use for each of these applications.



Conclusions (2)

- Our empirical study of FSS performance shows that, despite key generation and update operations which are significantly more expensive than non-forward-secure equivalents, the performance of FSS is actually quite competitive if correctly used.
 - In environments such as a Certificate Authority, the overhead of FSS is almost nonexistent when costs are amortized.
- The greatest difference in performance between FSS and traditional signature schemes occurs when there are few signatures and verification made by each key; even in these cases, FSS performs only two to four times slower, not hundreds or thousands times slower as the raw operation costs might indicate.
 - These results show concretely that forward-secure signatures are very practical.
 - Many applications which currently use traditional signatures could be switched to using forward-secure signatures with little impact on performance, but an enormous impact on the amount of inconvenience faced on key exposure.
- To further help the adoption of forward-secure signatures, the reference implementation used in this study will be released under an open-source license and available, along with developer documentation, from the project web site at <http://anonymized/>.



Future Work

- There are several directions for future work based on this study.
- We have only looked at the performance of these generic FSS constructions using software implementations of the base signature schemes.
- Many applications are now relying on hardware-based cryptographic co-processors when making traditional signatures, and the role these devices play in FSS needs to be explored.
- Our comparison also focused only on generic constructions due to their desirable property of being built upon well known and well trusted traditional signatures.
- Nonetheless, the performance of the other schemes described in Section 2 such as Bellare-Miner and Itkis-Reyzin are also of interest, and in the future we hope to expand our FSS reference implementation to include these schemes.

