RSA Laboratories

Volume 5, No. 2, Summer/Fall 2002

CryptoBytes

CONTENTS

I. The TESLA Broadcast Authentication Protocol

Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song

ABSTRACT

One of the main challenges of securing broadcast communication is source authentication, or enabling receivers of broadcast data to verify that the received data really originates from the claimed source and was not modified en route. This problem is complicated by mutually untrusted receivers and unreliable communication environments where the sender does not retransmit lost packets. The authors present TESLA (Timed Efficient Stream Loss-tolerant Authentication), an efficient broadcast authentication protocol with low communication and computation overhead. TESLA scales to large numbers of receivers and tolerates packet loss. It is based on loose time synchronization between the sender and the receivers, and uses only symmetric cryptographic functions. Still, and as is demonstrated herein, TESLA has possible PKI applications.

II. Forward-Secure Signatures with Optimal Signing and Verifying

Gene Itkis and Leonid Reyzin

ABSTRACT

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 by Anderson and formalized by Bellare and Miner to address this weakness. The authors describe the concept of forward security, and introduce the first forward-secure signature scheme for which both signing and verifying are as efficient the Guillou-Quisquater signature scheme, one of the most efficient ordinary signature schemes. Signing and verifying signatures in their scheme requires just two modular exponentiations with a short exponent.

III. Attacks on RC4 and WEP

Scott Fluhrer, Itsik Mantin, and Adi Shamir

ABSTRACT

RC4 is the most widely used stream cipher in software applications. In this paper, the authors summarize the known attacks on RC4, and show that it is completely insecure in the natural mode of operation, which is used in the widely deployed Wired Equivalent Privacy protocol (WEP, which is part of the 802.11b Wi-Fi standard). They describe a new, passive ciphertext-only attack that can find an arbitrarily long key in a negligible amount of time that grows _linearly (rather than exponentially) with the key size. The attack becomes even faster if WEP is replaced by its proposed successor WEP2.*

*Editor's Note

RSA Laboratories continues to collaborate on improved protocols for wireless security. Fast Packet Keying is the current WEP successor in the short term, and AES-CCM is the proposed successor in the long term. For more details, visit
http://www.rsasecurity.com/rsalabs/technotes/wep-fix.html and
http://csrc.nist.gov/encryption/modes/proposedmodes/.

RSA LABORATORIES

The TESLA Broadcast Authentication Protocol*

Adrian Perrig Ran Canetti J. D. Tygar Dawn Song

Abstract

One of the main challenges of securing broadcast communication is source authentication, or enabling receivers of broadcast data to verify that the received data really originates from the claimed source and was not modified en route. This problem is complicated by mutually untrusted receivers and unreliable communication environments where the sender does not retransmit lost packets.

This article presents the TESLA (Timed Efficient Stream Loss-tolerant Authentication) broadcast authentication protocol, an efficient protocol with low communication and computation overhead, which scales to large numbers of receivers, and tolerates packet loss. TESLA is based on loose time synchronization between the sender and the receivers.

Despite using purely symmetric cryptographic functions (MAC functions), TESLA achieves asymmetric properties. We discuss a PKI application based purely on TESLA, assuming that all network nodes are loosely time synchronized.

1 Introduction

Broadcast communication is gaining popularity for efficient and large-scale data dissemination. Examples of broadcast distribution networks are satellite broadcasts, wireless radio broadcast, or IP multicast. While many broadcast networks can efficiently distribute data to multiple receivers, they often also allow a malicious user to impersonate the sender and inject broadcast packets — we call this a packet injection attack. (Source-Specific Multicast (SSM, EXPRESS) is a notable exception, and attempts to prevent this attack [17, 40].)

Because malicious packet injection is easy in many broadcast networks, the receivers want to ensure that the broadcast packets they receive really originate from the claimed source. A broadcast authentication protocol enables the receivers to verify that a received packet was really sent by the claimed sender.

Simply deploying the standard point-to-point authentication mechanism (i.e., appending a message authentication code (MAC) to each packet, computed using a shared secret key) does not provide secure broadcast authentication. The problem is that any receiver with the secret key can forge data and impersonate the sender. Consequently, it is natural to look for solutions based on asymmetric cryptography to prevent this attack; a digital signature scheme is an example of an asymmetric cryptographic protocol. Indeed, signing each data packet provides secure broad-

*Most of this work was done at UC Berkeley and IBM Research. The authors can be reached at adrian+@cs.cmu.edu, canetti@watson.ibm.com, tygar@cs.berkeley.edu, skyxd@cs.cmu.edu.

cast authentication; however, it has high overhead, both in terms of the time required to sign and verify, and in terms of the bandwidth. Several schemes were proposed that mitigate this overhead by amortizing a single signature over several packets, e.g., [14, 25, 28, 33, 38, 39]. However, none of these schemes is fully satisfactory in terms of bandwidth overhead, processing time, scalability, robustness to denial-of-service attacks, and robustness to packet loss. Even though some schemes amortize a digital signature over multiple data packets, a serious denial-of-service attack is usually possible where an attacker floods the receiver with bogus packets supposedly containing a signature. Since signature verification is often computationally expensive, the receiver is overwhelmed verifying bogus signatures.

Researchers proposed information-theoretically secure broadcast authentication mechanisms [10, 11, 12, 13, 20, 34, 35, 36]. These protocols have a high overhead in large groups with many receivers.

Canetti et al. construct a broadcast authentication protocol based on k different keys to authenticate every message with k different MAC's [7]. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Boneh, Durfee, and Franklin show that one cannot build a compact collusion resistant broadcast authentication protocol without relying on digital signatures or on time synchronization [4]. They show that any secure broadcast authentication protocol with per-packet overhead slightly less than the number of receivers can be converted into a signature scheme.

Another approach to providing broadcast authentication uses only symmetric cryptography, more specifically on message authentication codes (MACs), and is based on delayed disclosure of keys by the sender. This technique was independently discovered by Cheung [8] in the context of authenticating link state routing updates. A related approach was used in the Guy Fawkes protocol for interactive unicast communication [1]. In the context of multicast streamed data it was proposed by several authors [2, 3, 5, 27, 28].

The main idea of TESLA is that the sender attaches to each packet a MAC computed with a key k known only to itself. The receiver buffers the received packet without being able to authenticate it. A short while later, the sender discloses k and the receiver is able to authenticate the packet. Consequently, a single MAC per packet suffices to provide broadcast authentication, provided that the receiver has synchronized its clock with the sender ahead of time.

This article is an overview of the TESLA broadcast authentication protocol. A more detailed description is in a forthcoming book [30] and in our earlier publications [27, 28]. A standardization effort for TESLA is under way in the Multicast Security (MSEC) working group of the IETF [26]. TESLA is used in a wide variety of applications, ranging from broadcast authentication in sensor networks [29], to authentication of messages in ad hoc network routing protocols [18].

2 Background and Assumptions

TESLA requires that the receivers are loosely time synchronized with the sender. In this section, we review a simple protocol to achieve this time synchronization. TESLA also needs an efficient mechanism to authenticate keys at the receiver — we first review one-way chains for this purpose.

2.1 One-Way Chains

Many protocols need to commit to a sequence of random values. For this purpose, we repeatedly use a one-way hash function to generate a one-way chain. One-way chains are a widely-used cryptographic primitive. One of the first uses of one-way chains was for one-time passwords by Lamport [21]. Haller later used the same approach for the S/KEY one-time password system [16]. One-way chains are also used in many other applications.

Figure 1 shows the one-way chain construction. To generate a chain of length ℓ we randomly pick the last element of the chain s_ℓ. We generate the chain by repeatedly applying a one-way function F. Finally, s_0 is a commitment to the entire one-way chain, and we can verify any element of the chain through s_0, e.g. to verify that element s_i is indeed the element with index i of the hash chain, we check that $F^i(s_i) = s_0$. More generally, s_i commits to s_j if $i < j$ (to verify that s_j is part of the chain if we know that s_i is the ith element of the chain, we check that $F^{j-i}(s_j) = s_i$). We reveal the elements of the chain in this order $s_0, s_1, \ldots, s_{\ell-1}, s_\ell$. How can we store this chain? We can either create it all at once and store each element of the chain, or we can just store s_ℓ and compute any other element on demand. In practice, a hybrid approach helps to reduce storage with a small recomputation penalty. Jakobsson [19], and Coppersmith and Jakobsson [9] propose a storage efficient mechanism for one-way chains: a one-way chain with N elements only requires $\log(N)$ storage and $\log(N)$ computation to access an element.

In TESLA, the elements of the one-way chain are keys, so we call the chain a one-way key chain. Furthermore, any key of the one-way key chain commits to all following keys, so we call such a key a one-way key chain commitment, or simply key chain commitment.

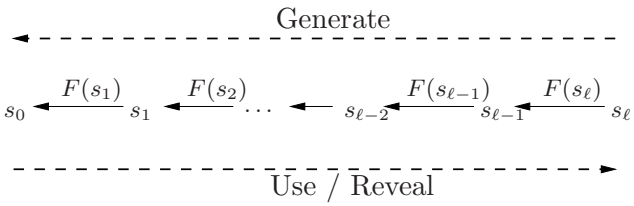

Figure 1: One-way chain example. The sender generates this chain by randomly selecting s_ℓ and repeatedly applying the one-way function F. The sender then reveals the values in the opposite order.

2.2 Time Synchronization

TESLA does not need the strong time synchronization properties that sophisticated time synchronization protocols provide [22, 24, 37], but only requires loose time synchronization, and that the receiver knows an upper bound on the sender's local time. We now outline a simple and secure time synchronization protocol that achieves this requirement. For simplicity, we assume the clock drift of both sender and receiver is negligible (otherwise the receiver can periodically resynchronize the time with the sender). We denote the real difference between the sender and the receiver's time with δ. In loose time synchronization, the receiver does not need to know the exact δ but only an upper bound on it, Δ, which we also refer to as the maximum time synchronization error.

We now describe a simple protocol for time synchronization, where each receiver performs explicit time synchronization with the sender. This approach does not require any extra infrastructure to perform time synchronization. We present a simple two-round time synchronization protocol that satisfies the requirement for TESLA, which is that the receiver knows an upper bound on the sender's clock. Reiter previously describes this protocol [31, 32].

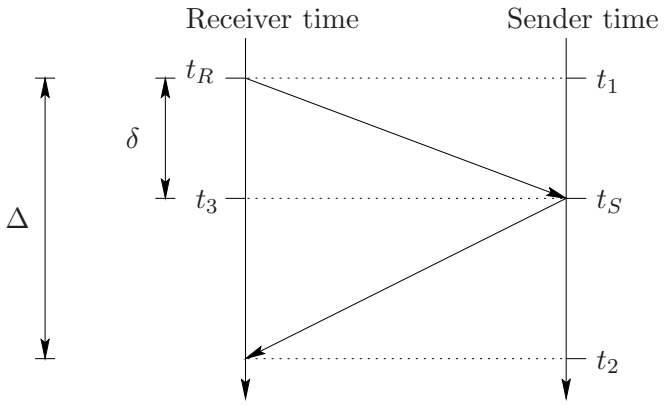

Figure 2: Direct time synchronization between the sender and the receiver. The receiver issues a time synchronization request at time t_R, at which time the sender's clock is at time t_1. The sender responds to the request at its local time t_S. In TESLA, the receiver is only interested in an upper bound on the sender's time. When the receiver has its current time t_r, it computes the upper bound on the current sender's time as $t_s \leq t_r - t_R + t_S$. The real synchronization error after this protocol is δ. The receiver, however, does not know the propagation delay of the time synchronization request packet, so it must assume that the time synchronization error is Δ (or the full round-trip time (RTT)).

Figure 2 shows a sample time synchronization between the receiver and the sender. In the protocol, the receiver first records its local time t_R and sends a time synchronization request containing a nonce to the sender.[1] Upon receiving the time synchronization request, the sender records its local time t_S and replies with a signed response packet containing t_S and the nonce.[2]

1. Setup. The sender S has a digital signature key pair, with the private key K_S^{-1} and the public key K_S. We assume a mechanism that allows a receiver R to learn the authenticated public key K_S. The receiver chooses a random and unpredictable nonce.

2. Protocol steps. Before sending the first message, the receiver records its local time t_R.

$$R \rightarrow S : \mathsf{Nonce}$$
$$S \rightarrow R : \{\mathsf{Sender\ time}\ t_S, \mathsf{Nonce}\}_{K_S^{-1}}$$

To verify the return message, the receiver verifies the digital signature and checks that the nonce in the packet equals the nonce it randomly generated. If the message is authentic, the receiver stores t_R and t_S. To compute the upper bound on the sender's clock at local time t, the receiver computes $t - t_R + t_S$.

Upon receiving the signed response, the receiver checks the validity of the signature and verifies that the nonce in the response packet equals the nonce in the request packet. If all verifications are successful, the receiver uses t_R and t_S to compute the upper bound of the sender's time: when the receiver has the current time t_r, it computes the upper bound on the current sender's time as $t_s \leq t_r - t_R + t_S$. The real synchronization error after this protocol is δ, as Figure 2 shows. The receiver, however, does not know the propagation delay of the time synchronization request packet, so it must assume that the time synchronization error is Δ (or the full round-trip time (RTT)).

[1] The security of this time synchronization protocol relies on the unpredictability of the nonce — if an attacker could predict the receiver's nonce, it could send a time synchronization request to the sender with that nonce, and replay the response later to the receiver.

[2] Interestingly, the processing and propagation delay of the response message does not change δ (assuming that the sender immediately records and replies with the arrival time of the request packet), since the receiver is only interested in an upper bound on the sender's clock. If the receiver were interested in the lower bound on the sender's clock, the processing delay and delay of the response message would matter. For more details on this refer to the more detailed time synchronization description [30].

A digital signature operation is computationally expensive, and we need to be careful about denial-of-service attacks in which an attacker floods the sender with time synchronization requests. Another problem is request implosion: the sender is overwhelmed with time synchronization requests from receivers. We address these issues in our earlier paper [27].

3 The TESLA Broadcast Authentication Protocol

A viable broadcast authentication protocol has the following requirements:

- Low computation overhead for generation and verification of authentication information.
- Low communication overhead.
- Limited buffering required for the sender and the receiver, hence timely authentication for each individual packet.
- Robustness to packet loss.
- Scales to a large number of receivers.

The TESLA protocol meets all these requirements with low cost — and it has the following special requirements:

- The sender and the receivers must be at least loosely time-synchronized as outlined in Section .
- Either the receiver or the sender must buffer some messages.

Despite the buffering, TESLA has a low authentication delay. In typical configurations, the authentication delay is on the order of one round-trip delay between the sender and receiver.

3.1 Sketch of TESLA protocol

We first outline the main ideas behind TESLA. Broadcast authentication requires a source of asymmetry, such that the receivers can only verify the authentication information, but not generate valid authentication information. TESLA uses time for asymmetry. We assume that receivers are all loosely time synchronized with the sender — up to some time synchronization error Δ, all parties agree on the current time. Here is a sketch of the basic approach:

- The sender splits up the time into time intervals of uniform duration. Next, the sender forms a one-way chain of self-authenticating values, and assigns the values sequentially to the time intervals (one key per time interval). The one-way chain is used in the reverse order of generation, so any value of a time interval can be used to derive values of previous time intervals. The sender defines a disclosure time for one-way chain values, usually on the order of a few time intervals. The sender publishes the value after the disclosure time.
- The sender attaches a MAC to each packet. The MAC is computed over the contents of the packet. For each packet, the sender determines the time interval and uses the corresponding value from the one-way chain as a cryptographic key to compute the MAC. Along with the packet, the sender also sends the most recent one-way chain value that it can disclose.

- Each receiver that receives the packet performs the following operation. It knows the schedule for disclosing keys and, since the clocks are loosely synchronized, can check that the key used to compute the MAC is still secret by determining that the sender could not have yet reached the time interval for disclosing it. If the MAC key is still secret, then the receiver buffers the packet.

- Each receiver also checks that the disclosed key is correct (using self-authentication and previously released keys) and then checks the correctness of the MAC of buffered packets that were sent in the time interval of the disclosed key. If the MAC is correct, the receiver accepts the packet.

One-way chains have the property that if intermediate values of the one-way chain are lost, they can be recomputed using later values. So, even if some disclosed keys are lost, a receiver can recover the key chain and check the correctness of packets.

The sender distributes a stream of messages $\{M_i\}$, and the sender sends each message M_i in a network packet P_i along with authentication information. The broadcast channel may be lossy, but the sender does not retransmit lost packets. Despite packet loss, each receiver needs to authenticate all the messages it receives.

We now describe the stages of the basic TESLA protocol in this order: sender setup, receiver bootstrap, sender transmission of authenticated broadcast messages, and receiver authentication of broadcast messages.

3.2 Sender Setup

TESLA uses self-authenticating one-way chains. The sender divides the time into uniform intervals of duration T_{int}. Time interval 0 will start at time T_0, time interval 1 at time $T_1 = T_0 + T_{int}$, etc. The sender assigns one key from the one-way chain to each time interval in sequence. The one-way chain is used in the reverse order of generation, so any value of a time interval can be used to derive values of previous time intervals.

The sender determines the length N of the one-way chain $K_0, K_1, \ldots, K_N$, and this length limits the maximum transmission duration before a new one-way chain must be created.[3] The sender picks a random value for K_N. Using a pseudo-random function f, the sender constructs the one-way function F: $F(k) = f_k(0)$. The remainder of the chain is computed recursively using $K_i = F(K_{i+1})$. Note that this gives us $K_i = F^{N-i}(K_N)$, so we can compute any value in the key chain from K_N even if we do not have intermediate values. Each key K_i will be active in time interval i.

3.3 Bootstrapping Receivers

Before a receiver can authenticate messages with TESLA, it needs to be loosely time synchronized with the sender, know the disclosure schedule of keys, and receive an authenticated key of the one-way key chain.

Various approaches exist for time synchronization [24, 37, 22]. TESLA, however, only requires loose time synchronization between the sender

[3] For details on how to handle broadcast streams of unbounded duration by switching one-way key chains, see [27]. For this article we assume that chains are sufficiently long for the duration of communication.

and the receivers, so a simple algorithm is sufficient. The time synchronization property that TESLA requires is that each receiver can place an upper bound of the sender's local time, as we discuss in Section .

The sender sends the key disclosure schedule by transmitting the following information to the receivers over an authenticated channel (either via a digitally signed broadcast message, or over unicast with each receiver):

- Time interval schedule: interval duration T_{int}, start time T_i and index of interval i, length of one-way key chain.
- Key disclosure delay d (number of intervals).
- A key commitment to the key chain K_i ($i < j - d$ where j is the current interval index).

3.4 Broadcasting Authenticated Messages

Each key in the one-way key chain corresponds to a time interval. Every time a sender broadcasts a message, it appends a MAC to the message, using the key corresponding to the current time interval. The key remains secret for the next $d-1$ intervals, so messages sent in interval j effectively disclose key K_{j-d}. We call d the key disclosure delay.

As a general rule, using the same key multiple times in different cryptographic operations is ill-advised — it may lead to cryptographic weaknesses. So we do not want to use key K_j both to derive key K_{j-1} and to compute MACs. Using a pseudo-random function family f', we construct the one-way function F': $F'(k) = f'_k(1)$. We use F' to derive the key to compute the MAC of messages: $K'_i = F'(K_i)$. Figure 3 depicts the one-way key chain construction and MAC key derivation. To broadcast message M_j in interval i the sender constructs packet $P_j = \{M_j \mid\mid \text{MAC}(K'_i, M_j) \mid\mid K_{i-d}\}$.

Figure 3 depicts the one-way key chain derivation, the MAC key derivation, the time intervals, and some sample packets that the sender broadcasts.

3.5 Authentication at Receiver

When a sender discloses a key, all parties potentially have access to that key. An adversary can create a bogus message and forge a MAC using the disclosed key. So as packets arrive, the receiver must verify that their MACs are based on safe keys: a safe key is one that is only known by the sender, and safe packets or safe messages have MACs computed with safe keys.

Receivers must discard any packet that is not safe, because it may have been forged.

We now explain TESLA authentication in detail: A sender sends packet P_j in interval i. When the receiver receives packet P_j, the receiver can use the self-authenticating key K_{i-d} disclosed in P_j to determine i. It then checks the latest possible time interval x the sender could currently be in (based on the loosely synchronized clock). If $x < i + d$ (recall that d is the key disclosure delay, or number of intervals that the key disclosure is delayed), then the packet is safe. The sender has thus not yet reached the interval where it discloses key K_i, the key that will verify packet P_j.

The receiver cannot yet verify the authenticity of packet P_j sent in interval i. Instead, it adds the triplet $(i, M_j, \text{MAC}(K'_i, M_j))$ to a buffer, and verifies the authenticity after it learns K'_i.

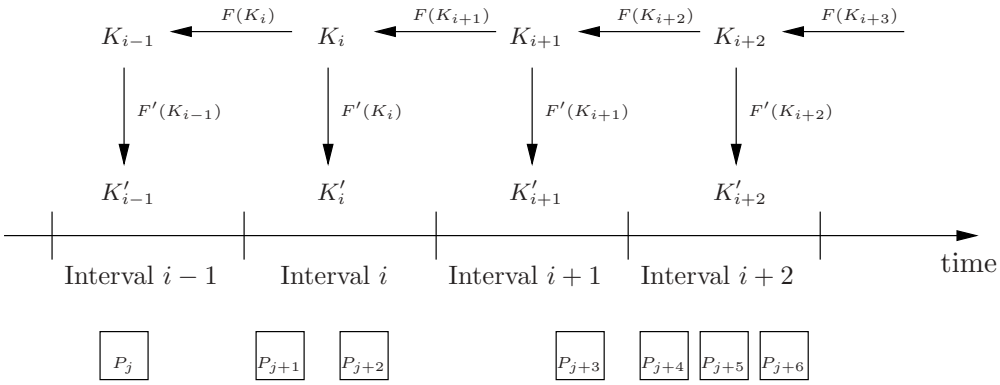

Figure 3: At the top of the figure is the one-way key chain (using the one-way function F), and the derived MAC keys (using the one-way function F'). Time advances left-to-right, and the time is split into time intervals of uniform duration. At the bottom of the figure, we can see the packets that the sender sends in each time interval. For each packet, the sender uses the key that corresponds to the time interval to compute the MAC of the packet. For example for packet P_{j+3}, the sender computes a MAC of the data using key K'_{i+1}. Assuming a key disclosure delay of two time intervals ($d = 2$), packet P_{j+3} would also carry key K_{i-1}.

What does a receiver do when it receives the disclosed key K_i? First, it checks whether it already knows K_i or a later key K_j ($j > i$). If K_i is the latest key received to date, the receiver checks the legitimacy of K_i by verifying, for some earlier key K_v ($v < i$) that $K_v = F^{i-v}(K_i)$. The receiver then computes $K'_i = F'(K_i)$ and verifies the authenticity of packets of interval i, and of previous intervals if the receiver did not yet receive the keys for these intervals (the receiver can derive them from K_i).

Note that the security of TESLA does not rely on any assumptions on network propagation delay, since each receiver locally determines the packet safety, i.e. whether the sender disclosed the corresponding key. However, if the key disclosure delay is not much longer than the network propagation delay, the receivers will find that the packets are not safe.

4 Discussion

4.1 TESLA Security Considerations

The security of TESLA relies on the following assumptions:

- The receiver's clock is time synchronized up to a maximum error of Δ. (We suggest that because of clock drift, the receiver periodically re-synchronizes its clock with the sender.)
- The functions F, F' are secure PRFs, and the function F furthermore provides weak collision resistance.[4]

As long as these assumptions are satisfied, it is computationally intractable for an attacker to forge a TESLA packet that the receivers will authenticate successfully.

[4] See our earlier paper for a formal security proof [28].

4.2 Achieving Asymmetric Security Properties with TESLA

Broadcast authentication requires an asymmetric primitive, which TESLA provides through loosely synchronized clocks and delayed key disclosure. TESLA shares many common properties with asymmetric cryptographic mechanisms. In fact, assuming that all nodes in a network are time synchronized, any key of the key chain serves as a key chain commitment and is similar to a public key of a digital signature: any loosely time synchronized receiver with an authentic key chain commitment can authenticate messages, but not forge a message with a MAC that receivers would accept.

We can construct an efficient PKI based solely on TESLA. Consider an environment with n communicating nodes. We assume that all nodes are loosely time synchronized, such that the maximum clock offset between any two nodes is Δ; and that all nodes know the authentic key chain commitment and key disclosure schedule of the certification authority (CA). We further assume that the CA knows the authentic key chain commitment and key disclosure schedule of every node. If a node A wants to start authenticating packets originating from another node B, A can contact the CA for B's key chain commitment and key disclosure schedule, which the CA sends authenticated with its TESLA instance. After the CA discloses the corresponding key, A can authenticate B's TESLA parameters and subsequently authenticate B's packets.

Note that TESLA is not a signature mechanism and does not provide non-repudiation, as anybody could forge "authentic" TESLA packets after the key is disclosed. However, in conjunction with a trusted time stamping mechanism, TESLA could achieve properties similar to a digital signature. Consider this setup: all nodes in the network are loosely time synchronized (as above with an upper bound on the synchronization error); and all nodes in the network trust the time stamping server [6, 15, 23]. The time stamping server timestamps all TESLA packets it receives. The time stamping server can broadcast the hooks to the trust chain authenticated with its TESLA instance. A judge who wants to verify that a sender sent packet P performs the following operations:

1. Receive the current value of the time stamping server's trust chain, ensure that it is safe, and wait for the TESLA key to authenticate it.

2. Based on the trust chain value, verify that packet P is part of the trust chain.

3. Verify that packet P was safe when the time stamping server received it (not necessary if the time stamping server only timestamps safe packets).

4. Retrieve key from the sender and verify it using the key chain commitment and disclosure schedule recorded by the time stamping server.

5. Verify that the authenticity of the packet, which implies that the correct sender must have generated the packet.

TESLA and a time stamping server can thus achieve non-repudiation. This example also shows that the TESLA authentication can also be performed after the key is already disclosed, as long as the verifier can check that the packet arrived safely.

5 Acknowledgments

We gratefully acknowledge funding support for this research. This research was sponsored in part the United States Postal Service (contract USPS 102592-01-Z-0236), by the United States Defense Advanced Research Projects Agency (contract N66001-99-2-8913), and by the United States National Science Foundation (grants 99-79852 and 01-22599). DARPA Contract N66001-99-2-8913 is under the supervision of the Space and Naval Warfare Systems Center, San Diego.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of the United States government, of DARPA, NSF, USPS, any of its agencies.

References

[1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. ACM Operating Systems Review, 32(4):9–20, October 1998.

[2] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, volume 2012 of Lecture Notes in Computer Science, pages 144–157, August 2000.

[3] F. Bergadano, D. Cavalino, and B. Crispo. Individual single source authentication on the mbone. In ICME 2000, Aug 2000.

[4] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In Advances in Cryptology — EUROCRYPT '2001, volume 2045 of Lecture Notes in Computer Science, pages 434–450, 2001.

[5] B. Briscoe. FLAMeS: Fast, Loss-Tolerant Authentication of Multicast Streams. Technical report, BT Research, 2000. http://www.labs.bt.com/people/briscorj/papers.html.

[6] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. In Advances in Cryptology — CRYPTO '98, volume 1462 of Lecture Notes in Computer Science, pages 486–501, 1998.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In INFOCOMM'99, pages 708–716, March 1999.

[8] S. Cheung. An efficient message authentication scheme for link state routing. In 13th Annual Computer Security Applications Conference, pages 90–98, 1997.

[9] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In Proceedings of the Fourth Conference on Financial Cryptography (FC '02), Lecture Notes in Computer Science, 2002.

[10] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In Advances in Cryptology — CRYPTO '91, volume 576 of Lecture Notes in Computer Science, pages 457–469, 1992.

[11] Y. Desmedt, Y. Frankel, and M. Yung. Multi-receiver / multi-sender network security: Efficient authenticated multicast / feedback. In Proceedings IEEE Infocom '92, pages 2045–2054, 1992.

[12] Y. Desmedt and M. Yung. Arbitrated unconditionally secure authentication can be unconditionally protected against arbiter's attacks. In Advances in Cryptology — CRYPTO '90, volume 537 of Lecture Notes in Computer Science, pages 177–188, 1991.

[13] F. Fujii, W. Kachen, and K. Kurosawa. Combinatorial bounds and design of broadcast authentication. IEICE Transactions, E79-A(4):502–506, 1996.

[14] R. Gennaro and P. Rohatgi. How to sign digital streams. In Advances in Cryptology — CRYPTO '97, volume 1294 of Lecture Notes in Computer Science, pages 180–197, 1997.

[15] S. Haber and W. Stornetta. How to timestamp a digital document. In Advances in Cryptology — CRYPTO '90, volume 537 of Lecture Notes in Computer Science, pages 437–455, 1991.

[16] N. Haller. The S/Key one-time password system. In Proceedings of the Symposium on Network and Distributed Systems Security, pages 151–157. Internet Society, February 1994.

[17] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In Proceedings of ACM SIGCOMM '99, September 1999.

[18] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (Mobicom 2002), September 2002. To appear.

[19] 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.

[20] K. Kurosawa and S. Obana. Characterization of (k,n) multi-receiver authentication. In Proceedings of the 2nd Australasian Conference on Information Security and Privacy (ACISP '97), volume 1270 of Lecture Notes in Computer Science, pages 205–215, 1997.

[21] L. Lamport. Password authentication with insecure communication. Communications of the ACM, 24(11):770–772, November 1981.

[22] L. Lamport and P. Melliar-Smith. Synchronizing clocks in the presence of faults. Journal of the ACM, 32(1):52–78, 1985.

[23] H. Lipmaa. Secure and Efficient Time-Stamping Systems. PhD thesis, Department of Mathematics, University of Tartu, Estonia, April 1999.

[24] D. Mills. Network Time Protocol (version 3) specification, implementation and analysis. Internet Request for Comment RFC 1305, Internet Engineering Task Force, March 1992.

[25] S. Miner and J. Staddon. Graph-based authentication of digital streams. In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 232–246, May 2001.

[26] Multicast security ietf working group (msec). http://www.ietf.org/html.charters/msec-charter.html, 2002.

[27] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2001), pages 35–46. Internet Society, February 2001.

[28] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signature of multicast streams over lossy channels.

In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 56–73, May 2000.

[29] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In Proceedings of Seventh Annual International Conference on Mobile Computing and Networks (Mobicom 2001), pages 189–199, 2001.

[30] A. Perrig and J. D. Tygar. Security Protocols for Broadcast Networks. Kluwer Academic Publishers, 2002. To appear.

[31] M. Reiter. A security architecture for fault-tolerant systems. PhD thesis, Department of Computer Science, Cornell University, August 1993.

[32] M. Reiter, K. Birman, and R. van Renesse. A security architecture for fault-tolerant systems. ACM Transactions on Computer Systems, 12(4):340–371, November 1994.

[33] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In Proceedings of the 6th ACM Conference on Computer and Communications Security, pages 93–100, November 1999.

[34] R. Safavi-Naini and H. Wang. New results on multireceiver authentication codes. In Advances in Cryptology — EUROCRYPT '98, volume 1403 of Lecture Notes in Computer Science, pages 527–541, 1998.

[35] R. Safavi-Naini and H. Wang. Multireceiver authentication codes: Models, bounds, constructions and extensions. Information and Computation, 151(1/2):148–172, 1999.

[36] G. Simmons. A cartesian product construction for unconditionally secure authentication codes that permit arbitration. Journal of Cryptology, 2(2):77–104, 1990.

[37] B. Simons, J. Lundelius-Welch, and N. Lynch. An overview of clock synchronization. In B. Simons and A. Spector, editors, Fault-Tolerant Distributed Computing, number 448 in LNCS, pages 84–96, 1990.

[38] D. Song, D. Zuckerman, and J. D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 258–270, May 2002.

[39] C. Wong and S. Lam. Digital signatures for flows and multicasts. In IEEE ICNP '98, 1998.

[40] Source-Specific Multicast IETF working group (SSM). http://www.ietf.org/html.charters/ssm-charter.html, 2002.

Forward-Secure Signatures with Optimal Signing and Verifying

Gene Itkis Leonid Reyzin
Boston University Computer Science Dept.
111 Cummington St.
Boston, MA 02215, USA
{itkis,reyzin}@bu.edu

Abstract

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 by Anderson [4] and formalized by Bellare and Miner [8] to address this weakness.

We describe the concept of forward security, as well as the first forward-secure signature scheme for which both signing and verifying are as efficient as for one of the most efficient ordinary signature schemes (Guillou-Quisquater [14]), each requiring just two modular exponentiations with a short exponent.

1 Introduction

Limitations of Digital 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 her and which by the imposter (who used the stolen key), there is no way for the verifier to distinguish them.

Thus upon such a secret key compromise, the signer should revoke her public key, and obtain a new one (with the corresponding non-compromised secret). But what to do with the already issued (before the compromise — in good faith) signatures. 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, she might even fake a compromise herself (for example, by anonymously posting her secret key on the internet and claiming to be the victim of a computer break-in).

Frequent Rekeying. One approach to preventing the theft of the secret key is to simply erase it—a securely erased key cannot be stolen. All the signatures produced with it cannot later become invalid. In other words, by erasing her secret key, the signer guarantees that the issued signatures remain trustworthy for the future (assuming that she is certain the key was not compromised prior to erasure).

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 withouth incurring the costs of changing public keys (and associated overhead). First proposed by Anderson in [4] and formalized by Bellare and Miner in [8], they enable the signer to frequently erase the secret key while maintaining the same public key. The notion of a forward-secure signature scheme is akin to the notion of "perfect forward secrecy" for key agreement [15], which protects past traffic even after long-term keys are compromised.[1]

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 component: the time period during which the signature is issued. If the time period is modified within a signature, it will no longer verify. Forward-security property means that even if the adversary obtains the current secret key, he cannot forge signatures for past time periods.

Applications. Consider, again, the example of a signer who issues digital credentials. Suppose she generated her keys on January 1, and her secret key was compormised on July 1. If she uses an ordinary signature scheme, then she has to revoke her old public key (effective retroactively, from January 1), and reissue (with a new key) all the credentials issued between January 1 and July 1. If, on the other hand, she uses a forward-secure scheme, then she merely has to revoke her public key from July 1 forward. The only credentials she needs to reissue with a new key are the ones issued on July 1 before the key compormise was discovered. Essentially, the potential liability she could incur from key compromise is greatly reduced by forward-secure signatures, from one year's worth of signatures to only one day's.

Now consider the example of a dishonest signer who wants to repudiate her signatures. Repudiation on the basis of key compromise must be allowed to protect the innocent signers whose keys get stolen, because no full-proof protection exists for secret keys (even tamper-resistant devices can be compromised or stolen) When ordinary signature schemes are used, this presents a potential liability for signature recipients for the duration of the key activity or longer—as long as the secret key is around.

It is, however, quite reasonable to require signers to securely erase their old keys. Thus, using a forward-secure signature scheme, potential liability for the signature recipients is reduced to only the current time period. In other words, if a signature is received on July 1, by July 2 the signer must have erased the secret key and can no longer claim that that key is stolen. It is reasonable, then, to shift the liability to the signer and to disallow repudiation on July 2: failing to securely erase the key is deemed the signer's fault. This, of course, assumes that the signer is expected to promptly detect key compromise: if she can reasonably demonstrate on July 2 that the key was stolen on July 1 without her knowledge, she should be able to repudiate the signature. This brings us to our next topic.

[1] The "forward" in "forward security" and "forward secrecy" is a constant source of debate. Some prefer "backward" since such schemes protect communications that happened before a key compromise. In our opinion, "forward" is justified, because if a communicaiton occurs when the key is secure, then it will be protected in the future no matter what happens with the key.

Implicit Assumptions. As seen above, secure erasure of the old secret keys (and all the intermediate results of any computations involving them) is crucial for all forward-secure schemes. As is well-known, this assumption may present serious challenges in the implementation: one has to make sure that no copies of anything sensitive are accidentally left unerased or are recoverable from hard disks and other storage devices ([10] addresses this problem).

Similarly, we relied on signer's ability to detect compromises. Just like in regular signature schemes, the sooner the determination is made and key revoked, the less damage the adversary can do. In particular, if forward-secure signatures are used to deter repudiation, the signer has to be provided with the means to detect key-compromise (otherwise, the signer can repudiate a signature by claiming she did not know that key compromise had occurred). This can be accomplished with the use of tamper-evident devices, which are much easier to build than tamper-proof ones (evidence of tampering may often be obtained from external indicators, such as broken-in office, a lost laptop, etc.)

Finally, we relied on the verifier's ability to check that the signature has the correct time period, and that the public key is valid during that time period. This assumes some synchronization between the signer and the verifier (and possibly the outside arbiter, if one exists). This assumption may present a problem for small devices without autonomous clocks, especially when the time period is short. If a signature is "post dated," (e.g., issued on July 1 but dated July 10), then the signer has more time to repudiate it based on a claim of key compromise (in the example above, 10 days).

We note that a time period need not actually correspond to physical time. For example, it can correspond to a particular number of signatures. A bank may require the signer to update the key after signing each check, to make sure that each individual check cannot be repudiated. Because the bank (presumably) knows how many checks have been issued, both parties can agree on what the correct "time" (in this case, check number) is.

Forward-Secure Schemes. Known forward-secure signature schemes can be divided into two categories: those that use arbitrary ordinary signature schemes in a black-box manner, and those that modify some specific signature scheme.

In the first category, the forward-secure schemes use some method in which a master public key is used to certify (perhaps via a chain of certificates or a Merkle tree) the current public key for a particular time period. Usually, these schemes require increases in storage space by noticeable factors in order to maintain the current (public) certificates and the (secret) keys for issuing future certificates. They also require longer verification times than ordinary signatures do, because the verifier needs to verify the entire certificate chain in addition to verifying the actual signature on the message. The first such scheme was proposed by Bellare and Miner [8]. Significant improvements were proposed by Krawczyk [21] and by Malkin, Micciancio and Miner [22]. The resulting schemes are quite competitive; exact comparison is made difficult by the multitude of implementation options available—most importantly, by the choice of the underlying ordinary signature scheme.

In the second category, there are four known schemes, each based on a different ordinary signature scheme: the scheme of Bellare and Miner [8] based on the Fiat-Shamir [11], the scheme of Abdalla and Reyzin [2] based the 2^t-th root [24, 25, 23], the scheme of Itkis and Reyzin [16] based on Guillou-Quisquater [14], and the scheme of Kozlov and Reyzin [20] based on the ideas of

[12, 6, 27]. Of these, the scheme of [16] has most efficient signing and verifying algorithms: each require just two exponentiations with a short exponent, just like the underlying ordinary signature scheme.[2] Moreover, it has short secret and public keys, and requires no additional storage. We therefore present it in detail below. More intuition and formal proofs can be found in [16].

Further Reading. The seminal paper on forward-secure signatures by Bellare and Miner [8] contains an excellent discussion as well as formal definitions of the concept. Other forward-secure signature schemes [21, 22, 20] may offer advantages depending on the relative importance of the speeds and sizes of various components. The work [3] contains a good exposition of techniques used to prove security of many forward-secure signatures. Some extensions of the forward-secure model are considered in [1, 27, 18]. Forward-secure encryption is addressed in [19].

2 Construction

2.1 Guillou-Quisquater Signatures

In [14], Guillou and Quisquater proposed the following three-round identification scheme. Let k and l be two security parameters (reasonable values ranges are 1024–2048 and 128–160, respectively). The prover's secret key consists of a k-bit modulus n (a product of two random primes p_1, p_2), an $(l+1)$-bit exponent e that is relatively prime to $\phi(n) = (p_1 - 1)(p_2 - 1)$, and a random $s \in Z_n^*$. The public key consists of n, e and v where $v = 1/s^e \bmod n$.

In the first round, the prover generates a random $r \in Z_n^*$, computes the commitment $y = r^e \pmod n$ and sends y to the the verifier. In the second round, the verifier sends a random l-bit challenge σ to the prover. In the third round, the prover computes and sends to the verifier $z = rs^\sigma$. To check, the verifier computes $y' = z^e v^\sigma$ and checks if $y = y' \pmod n$ (and $y \not\equiv 0 \pmod n$).

The standard transformation of Fiat and Shamir [11] can be applied to this identification scheme to come up with the GQ signature scheme, presented in Figure 1. Essentially, the interactive verifier's l-bit challenge σ is now computed using a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^l$ (such as SHA-1) applied to the message M and the commitment y.

The scheme's security is based on the assumptions that computing roots modulo composite n is infeasible without knowledge of its factors, and that H acts like a "random oracle."

2.2 Making it Forward-Secure

Main Idea. We combine the GQ scheme with Shamir's [26] observation that roots of different degrees are essentially independent, as long as the degrees are relatively prime. In other words, instead of just using s (which is a root of $1/v$ of degree e), the signer will simply use roots of $1/v$ of different degrees in different time periods. It can be proven that knowing the secret for one time period does not help sign in another time period—this will be the basis for forward security.

More precisely, let $e_1, e_2, \ldots, e_T$ be distinct integers, all greater than 2^l, all pairwise relatively prime and relatively prime with $\phi(n)$. Let $s_1, s_2, \ldots, s_T$ be such that $s_i^{e_i} \equiv 1/v \pmod n$ for $1 \le i \le T$. In time period i, the signer will simply use the GQ scheme with the secret key (n, s_i, e_i) and the verifier will use the GQ scheme with the public key (n, v, e_i).

[2] This scheme was further extended to an intrusion-resilient scheme [17], which provides much stronger guarantees of protecting not only the past but also the future.

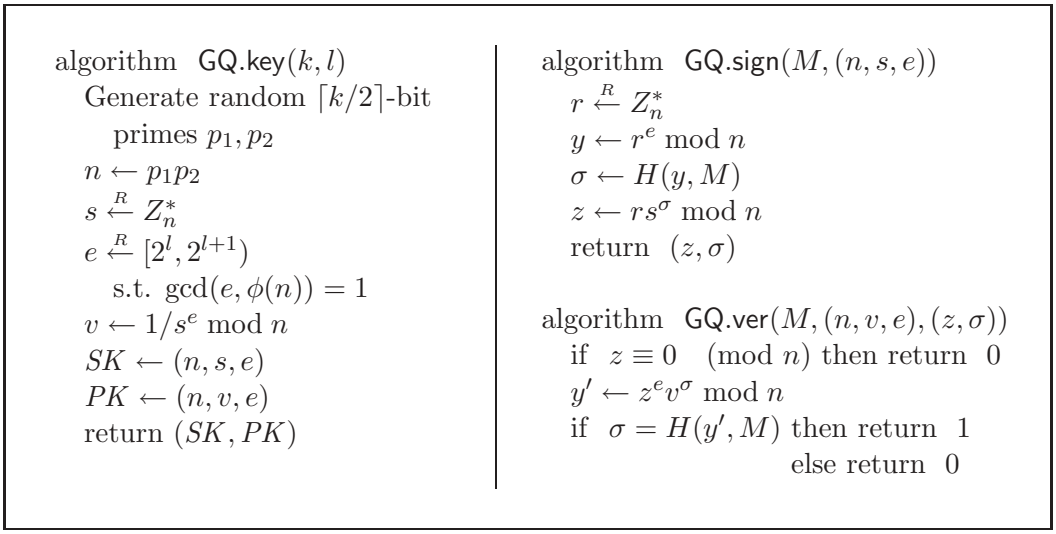

Figure 1: The GQ Signature Scheme

This idea is quite simple. However, we still need to address the following two issues: (i) how the signer computes the s_i's, and (ii) how both the signer and the verifier obtain the e_i's.

Computing s_i's. Notice that if the signer were required to store all the s_i's, this scheme would require secret storage that is linear in T. However, this problem can be easily resolved. Let $f_i = e_i \cdot e_{i+1} \cdot \ldots \cdot e_T$. Let $s_{[i,T]}$ be such that $s_{[i,T]}^{f_i} \equiv 1/v \pmod n$. During the j-th time period, the signer stores s_j and $s_{[j+1,T]}$. At update time, the signer computes $s_{j+1} = s_{[j+1,T]}^{f_{j+2}} \bmod n$ and $s_{[j+2,T]} = s_{[j+1,T]}^{e_{j+1}} \bmod n$. This allows secret storage that is independent of T: only two values modulo n are stored at any time (the f_i and e_i values are not stored—see below). It does, however, require computation linear in T at each update, because of the high cost of computing s_{j+1} from $s_{[j+1,T]}$.

We can reduce the computation at each update to be only logarithmic in T by properly utilizing precomputed powers of $s_{[j+1,T]}$. This will require us, however, to store $1 + \log_2 T$ secrets instead of just two. This optimization concerns only the efficiency of the update algorithm and affects neither the other components of the scheme nor the security, and is therefore presented separately in Section .

Obtaining e_i's. In order for the scheme to be secure, the e_i's need to be relatively prime with each other and with $\phi(n)$, and greater than 2^l. Unlike in the GQ scheme, where the single value e is included in the public key, we do not want to include all the e_i values in the public key, because this will lengthen the key too much (by $T(l+1)$ bits). Therefore, we simply require the signer to include the correct e_i value in the signature. Although a forger may attempt to use an incorrect e_i value in a forgery, this is not a problem: it can be proven that forgery for time period i is impossible for any e value, as long as it is guaranteed to be relatively prime with the values $e_{i+1}, e_{i+2}, \ldots, e_T$.

The last conditions can be verified in a number of different ways. The simplest is to divide the entire space of $l+1$-bit integers into T intervals

(the i-th interval is the set of integers between $2^l(1+(i-1)/T)$ and $2^l(1+i/T)$)), and pick each e_i as prime from the i-th interval. Then the verifier simply needs to check that the alleged e_i in the signature is indeed in the correct interval—it is then guaranteed to be relatively prime with the values $e_{i+1}, e_{i+2}, \ldots, e_T$ (because they are primes greater than the alleged e_i value). Note that the verifier need not perform any primality testing.

The signer need not store all the e_i's. Only the one for the current time period is needed for signing. The ones for future periods are needed during the key update procedure only. They can be recovered as needed; the signer need simply store enough information to be able to recover them. In most applications, it is probably simplest to generate each e_i as the first prime after the beginning of the i-th interval, i.e., the first prime after $2^l(1+(i-1)/T)$. Then one can either regenerate e_i's from scratch, or, if some of storage is available, store the short offset between the beginning of the interval, $2^l(1+(i-1)/T$, and the e_i value itself. We note that as l is about 128, $(l+1)$-bit primes should be quick to generate.

The resulting scheme is presented in Figure 2. Note that, to ensure that each e_i is relatively prime with $\phi(n)$, we simply use "safe" primes as factors of n: i.e., primes p_1 and p_2 such that $p_1 = q_1 + 1$ and $p_2 = q_2 + 1$, where q_1 and q_2 are themselves prime. Then $\phi(n) = 4q_1q_2$, and has only two odd prime factors. One can relax the condition: instead of having p_1 and p_2 be safe primes, we can simply require that p_1-1 and p_2-1 have no odd prime factors of size less than 2^{l+1}. This will speed up key generation.

The scheme is provably forward-secure based on the so-called "strong RSA" assumption [7, 12]. The assumption is that it is hard to extract a root of v modulo n of any degree between 2 and 2^{l+1}). The proof, given in [16], is in the random oracle model [9]: it assumes that the hash function H behaves like a random function.

Other methods for obtaining e_i's. The following other methods for finding e_i's offer potential benefits to the signer at a small cost to the verifier.

If both the signer and the verifier have readily available access to a table of first T odd primes (which is particularly plausible if T is small), then each e_i can be simply picked as a power if the i-th prime. In other words, if we let ϵ_i be the i-th odd prime (starting our counting at $\epsilon_1 = 3$, $\epsilon_2 = 5, \ldots$), we can let $e_i = \epsilon_i^{\alpha_i}$, where α_i is a large enough value to ensure that $e_i > 2^l$ (for example, $\alpha_i = l\lfloor \log_2 \epsilon_i \rfloor$).

The algorithms would change as follows: the key generation and update algorithms would now generate e_i as $\epsilon_i^{\alpha_i}$, and would therefore to have to perform any primality testing on e_i's. The signing algorithm would remain the same, but would not need to output e_j as part of the signature, thus shortening the signature. The verification algorithm, instead of obtaining e_j from the signature and verifying that it is in the correct inverval, would simply have to compute e_j directly as $\epsilon_j^{\alpha_j}$.

Yet another method is available if the table of small primes is inconvenient for a particular application. Let S be some small integer value (to be explained shortly), and let ϵ_i be a first prime in the interval $[(i-1)S, iS)$. Again, let $e_i = \epsilon_i^{\alpha_i}$, where α_i is a large enough value to ensure that $e_i > 2^l$. S needs to be picked large enough to guarantee that there exists at least one prime in each interval $[(i-1)S, iS)$ for $1 \le i \le T$. For example, $S = 34$ will work for $T = 1024$, because the largest gap between the first 1024 primes is 34; by the same reasoning, $S = 72$ will work for $T = 10^4$, $S = 114$ will work for $T = 10^5$, and $S = 154$ will work for $T = 2^{20} > 10^6$.

algorithm $\mathsf{IR.key}(k, l, T)$

Generate random $\lceil k/2 \rceil$-bit safe primes p_1, p_2: generate primes q_1, q_2, s.t. $p_i \leftarrow 2q_i+1$ are prime

$n \leftarrow p_1 p_2$

$s_{[1,T]} \stackrel{R}{\leftarrow} Z_n^*$

Generate primes $e_1, \ldots, e_T$ s.t. $2^l(1 + (i-1)/T) \leq e_i < 2^l(1 + i/T)$.

$f_2 \leftarrow e_2 \cdot \ldots \cdot e_T \bmod \phi(n)$, where $\phi(n) = 4q_1q_2$

$s_1 \leftarrow s_{[1,T]}^{f_2} \bmod n$; $s_{[2,T]} \leftarrow s_{[1,T]}^{e_1} \bmod n$

$v \leftarrow 1/{s_1}^{e_1} \bmod n$

$SK_1 \leftarrow (n, s_1, s_{[2,T]}, e_1)$ (n and e_1 are included for convenience and need not be kept secret)

$PK \leftarrow (n, v, T)$

Securely erase all the intermediate results, leaving only SK_1, PK unerased.

In particular, $q_1, q_2, p_1, p_2, \phi(n)$ must be erased.

Note that the values n and $e_1, \ldots, e_T$ need not be kept secret or erased.

---

algorithm $\mathsf{IR.sign}(M, SK_j)$

Let $SK_j = (n, s_j, s_{[j+1,T]}, e_j)$

$r \stackrel{R}{\leftarrow} Z_n^*$

$y \leftarrow r^{e_j} \bmod n$

$\sigma \leftarrow H(y, M, j, e_j)$

$z \leftarrow r{s_j}^{\sigma} \bmod n$

return (z, σ, j, e_j)

algorithm $\mathsf{IR.ver}(M, PK, (z, \sigma, j, e))$

Let $PK = (n, v, T)$

if $e \geq 2^l(1 + j/T)$ or $e < 2^l$ or e is even then return 0

if $z \equiv 0 \pmod n$ then return 0

$y' \leftarrow z^e v^{\sigma} \bmod n$

if $\sigma = H(y', M, j, e)$ then return 1 else return 0

---

algorithm $\mathsf{IR.update}(SK_j)$

Let $SK_j = (n, s_j, s_{[j+1,T]}, e_j)$

if $j = T$ then return "error"

Recover $e_{j+1}, \ldots, e_T$

$s_{j+1} \leftarrow s_{[j+1,T]}^{e_{j+2} \cdot \ldots \cdot e_T} \bmod n$; $s_{[j+2,T]} \leftarrow s_{[j+1,T]}^{e_{j+1}} \bmod n$

$SK_{j+1} \leftarrow (n, s_{j+1}, s_{[j+2,T]}, e_{j+1})$

Securely erase SK_j and all the intermediate results. Leave only SK_{j+1} unerased.

Figure 2: Our forward-secure signature scheme (without efficiency improvements). Highlighted in red are the differences from the GQ (Fig. 1).

The algorithms would change as follows: the key generation and update algorithms would now first generate each ϵ_i as a prime from the interval $[(i-1)S, iS)$, and then compute e_i as $\epsilon_i^{\alpha_i}$. Thus, they would perform primality testing on the short value ϵ_i, rather than the long value e_i. The signing algorithm would remain the same, but would output ϵ_j as part of the signature, instead of e_j. The verification algorithm would obtain ϵ_j from the signature, check that it is odd and greater than $(j-1)S$, and compute e_j as $\epsilon_i^{\alpha_i}$. Note that, again, the verification algorithm would not need to perform primality testing.

3 Optimizing key update

The key update in our scheme requires computing s_i such that $s_i^{e_i} \equiv 1/v \bmod n$. The solution we used in the previous section took $T-i-1$ exponentiations. This computation can be reduced dramatically if the storage is increased slightly. Specifically, in this section we demonstrate how replacing the single secret $s_{[j,T]}$ with $\log_2 T$ secrets can reduce the complexity of the update algorithm to only $\log_2 T$ exponentiations.

Abstracting the Problem. Consider all subsets of $Z_T = \{1, 2, \ldots, T\}$. Let each such subset $S \subseteq Z_T$ correspond to the secret value $s_S = s_{[1,T]}^{\prod_{i \notin S} e_i}$. For example, $s_{[1,T]}$ corresponds to Z_T, $s_{[i,T]}$ corresponds to $\{i, i+1, \ldots, T\}$, v^{-1} corresponds to the empty set, and each s_i corresponds to the singleton set $\{i\}$. Raising some secret value s_S to power e_i corresponds to dropping i from S.

Thus, instead of secrets and the exponentiation operation, we can consider sets and the operation of removing an element. Our problem, then, can be reformulated as follows: design an algorithm that, given Z_T, outputs (one-by-one, in order) the singleton sets $\{i\}$ for $1 \le i \le T$. The only way to create new sets is to remove elements from known sets. The algorithm should minimize the number of element-removal operations (because they correspond to the expensive exponentiation operations).

Pebbling. Let each subset of Z_T correspond to a node in a graph. Connect two sets by a directed edge if the destination can be obtained from the source by dropping a single element. The resulting graph is the T-dimensional hypercube, with directions on the edges (going from higher-weight nodes to lower-weight nodes; see Fig. 3).

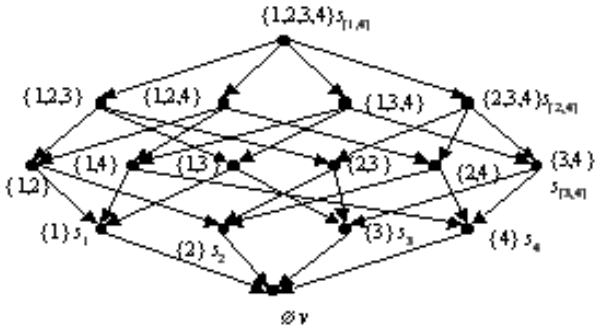

Figure 3: Hypercube for the subsets of $\{1, 2, 3, 4\}$.

Consider now ommitting some of the edges of this hypercube, together with the $\emptyset$ node corresponding to the value v. The resulting graph looks much simpler (see Fig. 4) and contains all the edges need for our algorithm.

Figure 4: Simplified tree on the non-empty subsets of $\{1, 2, 3, 4\}$.

Represent each set stored as a pebble at the corresponding node in a graph. Then removing an element from a set corresponds to moving the corresponding pebble down the corresponding directed edge. If the original set is preserved, then a "clone" of a pebble is left at the original node.

Our goal now can be reformulated as follows in terms of pebbles: find a pebbling strategy that, starting at the node Z_T, reaches every node $\{i\}$ in order. Moreover, we wish to optimize this strategy by minimizing the following three parameters:

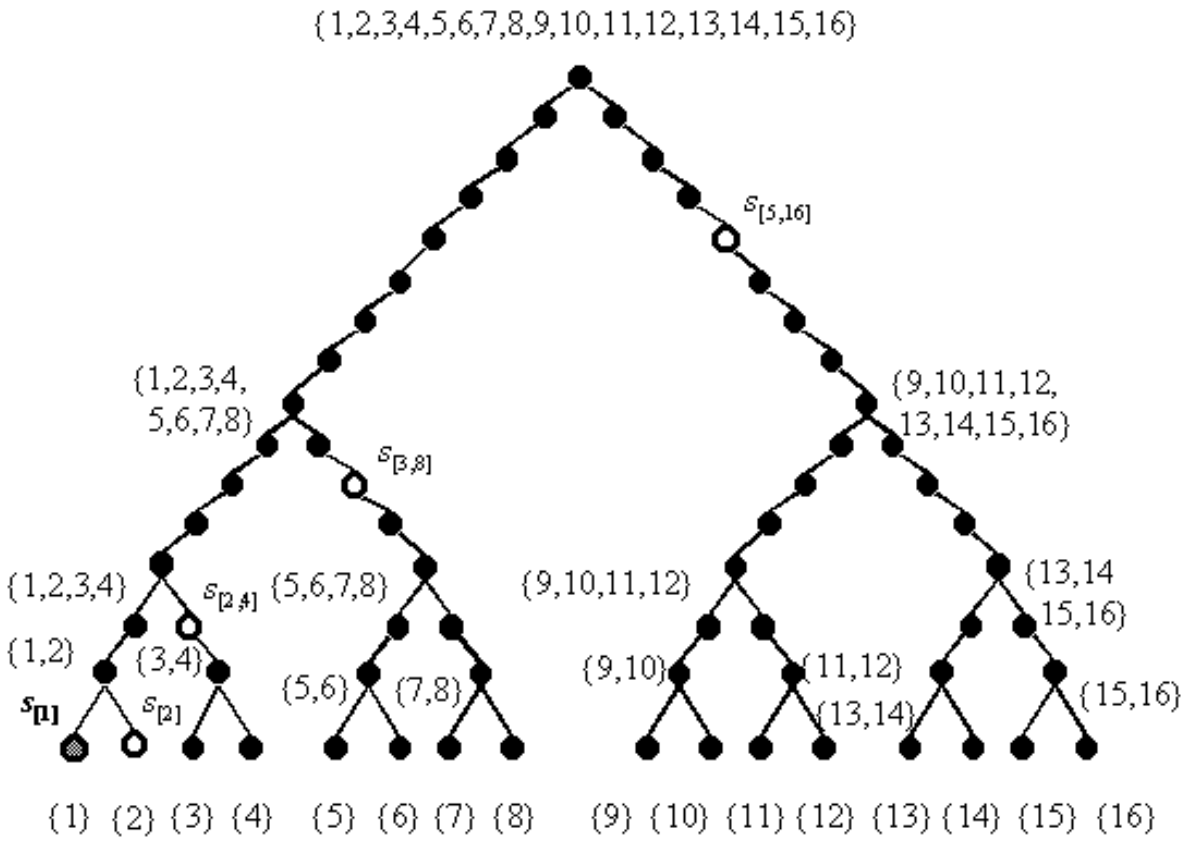

Figure 5: Pebbling the tree on the interval $\{1, ..., 16\}$. The configuration corresponds to the time when the first singleton $\{1\}$ is output. The pebbles are at the nodes $\{2\}; \{3, 4\}; \{4, 5, 6, 7, 8\}$; and $\{6, ..., 16\}$, and correspond to the four stored values $s_2, s_{[3,4]}, s_{[4,8]}$, and $s_{[6,16]}$.

(a) storage requirements — the number of pebbles used at any given time; (b) total number of exponentiations — the total number of pebble moves; and (c) the worst-case running time of the update algorithm — the max number of pebble moves between any two consecutive hits of a singleton.

The Pebbling Algorithm. Suppose we start with a single pebble at the root of the tree. Each pebble moves down the tree as follows. When reaching a branch (a node with two children) the pebble splits into two siblings — one taking the left edge and the other taking the right. The right sibling "donates" its moves to the left one, until the left sibling reaches the next branch. Then (when the left sibling splits again) the right pebble proceeds down its right path. A pebble is discarded upon reaching a leaf.

Thus, left-moving pebbles always move two edges per time period, while right-moving pebbles stay still while their left-moving pebbles are moving, and then move one-edge per time period until the next branch (their average speed thus is 2/3).

To use pebbling for key update, simply replace the $s_{[i+2,T]}$ value in the secret key with a list L of pebbles, generated and updated according to the detailed algorithms Figure 6. The secret s_{i+1} is generated during key update from L (instead of via long exponentiation from $s_{[i+2,T]}$).

Additional Steps for IR.key:
  Define data structure Pebble $= \{s, pb, pe, rb, re\}$
    // s is the secret that the pebble contains; the other four variables are named as follows:
    // "p" stands for "position," "r" for "responsibility," "b" for "begin" and "e" for "end"
  Let Pebble $R \leftarrow \{s_{[1,T]}, 1, 1, T, T\}$; let L be a list of Pebbles, containing the single element R
  for $i \leftarrow -\lfloor (T-2)/2 \rfloor$ to 0
    PebbleStep(L)

New Steps for IR.update (instead of modular exponentiations):
  PebbleStep(L)
  Remove the first element of L; its s value is s_{j+1} to be put into SK_{j+1}

Procedure PebbleStep(L)
  for each Pebble P in L, in order
    if $P.pb = P.pe$ do nothing // Pebble doesn't move because it is at destination
    else if $P.pb = P.rb$ then MoveLeft(P) // Move left twice, unless you hit bottom after first move
      if $P.pb \neq P.pe$ then MoveLeft(P) and skip over the next pebble in L
    else MoveRight(P)

Procedure MoveLeft(P)
  if $P.pe = P.re$ // Need to split this pebble into two
    Create a new Pebble $P1$, and insert it into L immediately following P
    $P1.s \leftarrow P.s, P1.pb \leftarrow P.pb, P1.pe \leftarrow P.pe, P1.re \leftarrow P.re, P1.rb \leftarrow \lfloor (P.rb + P.re + 1)/2 \rfloor$
    $P.re \leftarrow \lfloor (P.rb + P.re - 1)/2 \rfloor$
  $P.s \leftarrow P.s^{e_{pe}} \bmod n$; $P.pe \leftarrow P.pe - 1$

Procedure MoveRight(P)
  $P.s \leftarrow P.s^{e_{pb}} \bmod n$; $P.pb \leftarrow P.pb + 1$

Figure 6: Pebbling techniques to speed up key update

In [16] we prove that this strategy guarantees that at any moment there are at most $1 + \log_2 T$ pebbles and that all the pebbles always arrive at the leaves at the right times. The amount of steps in any time period does not exceed $\log_2 T$.

We also show that using pebbling does not compromise security, because none of the secrets contained in the pebbles can assist the adversary in forging past signatures (in fact, all the secrets in the pebbles at time period i can be derived from $s_{[i,T]}$).

References

[1] Michel Abdalla, Sara Miner, and Chanathip Namprempre. Forward-secure threshold signature schemes. In David Naccache, editor, Progress in Cryptology — CT-RSA 2001, volume 2020 of Lecture Notes in Computer Science, pages 143–158. Springer-Verlag, April 8-12 2001.

[2] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, Advances in Cryptology—ASIACRYPT 2000, volume 1976 of Lecture Notes in Computer Science, pages 116–129, Kyoto, Japan, 3–7 December 2000. Springer-Verlag. Full version available from the Cryptology ePrint Archive, record 2000/002, http://eprint.iacr.org/.

[3] Jee Hea An, Michel Abdalla, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Samir transform: Minimizing assumptions for security and forward-security. Available from http://www-cse.ucsd.edu/users/mihir/papers/id-sig.html.

[4] Ross Anderson. Invited lecture. Fourth Annual Conference on Computer and Communications Security, ACM; summary appears in [5], 1997.

[5] Ross Anderson. Two remarks on public key cryptology. http://www.cl.cam.ac.uk/users/rja14/, 2001.

[6] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, Advances in Cryptology—CRYPTO 2000, volume 1880 of Lecture Notes in Computer Science, pages 255–270. Springer-Verlag, 20–24 August 2000.

[7] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, Advances in Cryptology—EUROCRYPT 97, volume 1233 of Lecture Notes in Computer Science, pages 480–494. Springer-Verlag, 11–15 May 1997.

[8] Mihir Bellare and Sara Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, Advances in Cryptology—CRYPTO '99, volume 1666 of Lecture Notes in Computer Science, pages 431–448. Springer-Verlag, 15–19 August 1999. Revised version is available from http://www.cs.ucsd.edu/˜mihir/.

[9] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Proceedings of the 1st ACM Conference on Computer and Communication Security, pages 62–73, November 1993. Revised version appears in http://www.cs.ucsd.edu/˜mihir/.

[10] Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. How to forget a secret. In STACS '99: 16th Annual Symposium on Theoretical Aspects in Computer Science, volume 1563 of Lecture Notes in Computer Science. Springer-Verlag, March 1999.

[11] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, Advances in Cryptology—CRYPTO '86, volume 263 of Lecture Notes in Computer Science, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[12] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, Advances in Cryptology—CRYPTO '97, volume 1294 of Lecture Notes in Computer Science, Springer-Verlag, 1997.

[13] Shafi Goldwasser, editor. Advances in Cryptology—CRYPTO '88, volume 403 of Lecture Notes in Computer Science. Springer-Verlag, 1990, 21–25 August 1988.

[14] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Goldwasser [13], pages 216–231.

[15] C. Günther. An identity-based key-exchange protocol. In G. Brassard, editor, Advances in Cryptology—CRYPTO '89, volume 435 of Lecture Notes in Computer Science. Springer-Verlag, 1990, 20–24 August 1989.

[16] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, Advances in Cryptology—CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 332–354. Springer-Verlag, 19–23 August 2001.

[17] Gene Itkis and Leonid Reyzin. SIBIR: Signer-base intrusion-resilient signatures. In Yung [28]. Available from http://eprint.iacr.org/2002/054/.

[18] Gene Itkis and Leonid Reyzin. SIBIR: Signer-base intrusion-resilient signatures. In Yung [28]. Available from http://eprint.iacr.org/2002/054/.

[19] Jonathan Katz. A forward-secure public-key encryption scheme. Technical Report 2002/060, Cryptology e-print archive, http://eprint.iacr.org, 2002.

[20] Anton Kozlov and Leonid Reyzin. Forward-secure signatures with fast key update. Unpublished Manuscript, 2002.

[21] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In Seventh ACM Conference on Computer and Communication Security, 2000.

[22] Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Advances in Cryptology—EUROCRYPT 2002, Springer-Verlag, 2002.

[23] Silvio Micali. A secure and efficient digital signature algorithm. Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, Cambridge, MA, March 1994.

[24] Kazuo Ohta and Tatsuaki Okamoto. A modification of the Fiat-Shamir scheme. In Goldwasser [13], pages 232–243.

[25] Heidroon Ong and Claus P. Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In I. B. Damgård, editor, Advances in Cryptology—EUROCRYPT 90, volume 473 of Lecture Notes in Computer Science, pages 432–440. Springer-Verlag, 1991.

[26] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. ACM Transactions on Computer Systems, 1(1):38–44, February 1983.

[27] Dawn Xiaodon Song. Practical forward secure group signature schemes. In Eighth ACM Conference on Computer and Communication Security, pages 225–234. ACM, November 5–8 2001.

[28] Moti Yung, editor. Advances in Cryptology—CRYPTO 2002, Lecture Notes in Computer Science. Springer-Verlag, 18–22 August 2002.

Attacks On RC4 and WEP

Scott Fluhrer
Cisco Systems, Inc.,
170 West Tasman Drive,
San Jose, CA 95134,
sfluhrer@cisco.com

Itsik Mantin
Computer Science department,
The Weizmann Institute,
Rehovot 76100, Israel
itsik@wisdom.weizmann.ac.il

Adi Shamir
Computer Science department,
The Weizmann Institute,
Rehovot 76100, Israel.
shamir@wisdom.weizmann.ac.il

Abstract

RC4 is the most widely used stream cipher in software applications. In this paper we summarize the known attacks on RC4, and show that it is completely insecure in a natural mode of operation which is used in the widely deployed Wired Equivalent Privacy protocol (WEP, which is part of the 802.11b Wi-Fi standard). The new passive ciphertext-only attack can find an arbitrarily long key in negligible time which grows linearly (rather than exponentially) with the key size, and it gets even faster if WEP is replaced by its "improved" version WEP2.

1 The RC4 Stream Cipher

A large number of stream ciphers were proposed and implemented over the last twenty years. Most of these designs were based on linear feedback shift registers, which are easy to analyze, have good statistical properties, and are very efficient in hardware, but are relatively slow in software. There was a clear need for a different design which would be more software friendly, and in 1987 Ron Rivest developed the RC4 stream cipher, whose software implementation is extremely compact and efficient. Its design was kept a trade secret until 1994, when someone anonymously posted it to the Cypherpunks mailing list. RC4 is currently the most widely-used stream cipher in the world: It is used to protect Internet traffic using the SSL (Secure Sockets Layer) protocol, it is integrated into Microsoft Windows, Lotus Notes, Apple AOCE, Oracle Secure SQL, and many other software applications, and it was chosen to be part of the Cellular Digital Packet Data specification.

RC4 has a secret internal state which is a permutation S of all the $N = 2^n$ possible n-bit values, where n is typically chosen as 8. In addition, the state contains two indices $0 \leq i, j < N$. The initial state is derived from a key (whose typical size is between 40 and 256 bits) by a Key-Scheduling Algorithm (KSA), and then the pseudo-random generation algorithm (PRGA) alternately modifies the state (by exchanging two out of the N values) and produces an output (by picking one of the N values in S). For $n = 8$, this gives RC4 a huge state of about 1700 bits.

More specifically, the PRGA initializes i and j to 0, and then repeatedly increments i as a counter, increments j pseudo randomly (by adding the value of S pointed to by i), exchanges the two values of S pointed to by i and j, and outputs the value of S pointed to by $S[i] + S[j]$[1] (see Figure 1). Note that every entry of S is swapped at least once (possibly with itself) within any N consecutive rounds, and thus the permutation S

[1]Here and in the rest of this note all the numeric additions are carried out modulo N.

evolves fairly rapidly during the output generation process.

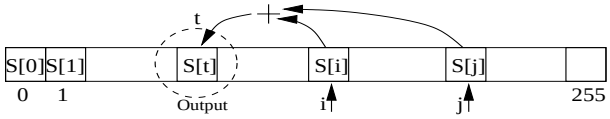

Figure 1: RC4 PRGA Round Operation

The KSA structure is very similar to the PRGA structure described above. It initializes S to the identity permutation and i and j to 0, and then applies N rounds in which i is sequentially stepped across S, and j is updated by adding to it $S[i]$ and the next word of the key (in cyclic order).

We denote the size of the key by ℓ, and the n-bit RC4 variant by RC4$_n$. The value in position p of the permutation S is denoted by $S[p]$. Using this notation, the RC4 algorithms are specified in Figure 2.

KSA(K)
Initialization:
  $S \leftarrow \langle 0, 1, \ldots, N-1 \rangle$
  $j \leftarrow 0$
Scrambling:
  For $i \leftarrow 0 \ldots N-1$
    $j \leftarrow j + S[i] + K[i \bmod \ell]$
    $S[i] \leftrightarrow S[j]$

PRGA(S)
Initialization:
  $i \leftarrow 0$
  $j \leftarrow 0$
Generation loop:
  $i \leftarrow i + 1$
  $j \leftarrow j + S[i]$
  $S[i] \leftrightarrow S[j]$
  $t \leftarrow S[i] + S[j]$
  Output $z \leftarrow S[t]$

Figure 2: The Key-Scheduling Algorithm and the Pseudo-Random Generation Algorithm

2 Security Overview of RC4

Since RC4 is such a widely used stream cipher, it had attracted considerable attention in the research community since it was first proposed. However, so far no one had found an attack on the PRGA part of RC4 which is even close to being practical: For $n = 8$ and sufficiently long keys, the best known attack requires more than 2^{700} time to find its initial state. However, many interesting properties of RC4 were found over the years. Due to space limitations, these properties are only briefly described in this note, and a more comprehensive overview of RC4 security, including a detailed description of all the published results, can be found in [Man01].

Finney specifies in [Fin94] a class of states that RC4 can never enter. This class contains all the states for which $i = a$, $j = a+1$ and $S[a+1] = 1$ (a fraction of $1/N^2$ of all the RC4 states are in this class). Analysis of states of this type indicates that these states are connected by short cycles of length $N(N-1)$ and thus this class is closed under the RC4 round operation. Since the state transition in RC4 is invertible and the initial state ($i = j = 0$) is not of this type, RC4 can never enter these states for any key. If RC4 could be initialized to such a state, the generated stream would have a very short period and its internal states could be elegantly extracted (as noted by Pudovkina and described in [Man01]).

The huge RC4 internal state is not random, and thus the size of the effective key does not properly reflect its security. A good benchmark for the security of RC4 variants is the time complexity of a "branch and bound" attack that eliminates large clusters of internal state candidates that contradict the generated outputs. Such an attack is analyzed in [MT98] and [KMP$^+$98], and is based on the "guess on demand" paradigm. These attacks simulate the generation process by keeping track of all the known permutation values and guessing/branching when an unknown value is needed in order to continue the simulation. The time complexity of this attack is analytically derived in [KMP$^+$98], and the results are compatible with experimental data obtained by scaled-down simulations. This attack is very inefficient (i.e., worse than exhaustive search for typical key sizes), and

can be carried out in practice only for $RC4_{n\leq 5}$ (e.g. for $n = 5$ its time complexity is 2^{53}, but for the common choice of $n = 8$ its time complexity is 2^{779}).

Mantin and Shamir use a variant of this attack in [MS01] (and more extensively in [Man01]). They present some interesting classes of RC4 states that make it possible to deduce several permutation values with high probability. Then they use the attack of Knudsen et al ([KMP+98]) in order to complete the rest of the internal state. This 2-phase attack is practical for the 5-bit version. More analysis of RC4 round operation is described in [MT98]. In addition, Jenkins in [Jen98] shows a significant correlation between the secret and known parts of the RC4 state. For example, this correlation implies that given i and z, the probability of $S[i]$ to be their difference $(i - z)$ is twice the expected $1/N$. This correlation is explained by Mantin in [Man01] as a special case of a more general correlation.

A significant research effort is dedicated to the statistical analysis of RC4 outputs, and in particular to the construction of distinguishers between RC4 and truly random bit generators. Golić describes in [Gol97] a linear statistical weakness of RC4, caused by a positive correlation between the second binary derivative of the LSB and 1. This weakness implies that RC4 outputs of length 2^{40} can be reliably distinguished from random strings. This result is further improved by Fluhrer and McGrew in [FM00]. They analyze the distribution of triplets consisting of the two outputs produced at times t and $t+1$, and the known value of $i \equiv t \pmod N$, and notice small biases in the distribution of $(7N-8)$ of these N^3 triplets: some of these probabilities are positively biased ($\frac{1}{N^3}(1+1/N)$), while others are negatively biased ($\frac{1}{N^3}(1-1/N)$). These biases enable distinguishing between RC4 and a truly random source by analyzing sequences of about 2^{30} output words.

Interesting as they are, all these statistical anomalies of the RC4 PRGA have little practical impact on its cryptographic strength, since they do not make it possible to derive either the plaintexts or the keys which correspond to given ciphertexts. However, the RC4 KSA is much more problematic, since it was designed to be unnecessarily simple: it initializes the secret S and j to known values, uses the key bytes in cyclic order to increment j, performs a single pass over S, and starts producing outputs immediately afterwards. As a result, the first few output bytes of the PRGA are either biased or related to some key bytes, and thus the analysis of these bytes makes it possible to attack RC4 in some modes of operation. The best protective mechanism against these attacks (which is used in some but not all implementations of RC4) is to discard a long prefix of PRGA outputs after completing the KSA part. Such dummy steps (which produce no outputs) are used in many other stream cipher designs, such as the A5/1 scheme which is used in GSM cellular phones and analyzed in [BSW00].

We now describe some of these prefix attacks on RC4 in more detail. Roos notes in [Roo95] that for keys which have $K[0] + K[1] = 0$, the first output is equal to $K[2] + 3$ with probability $2^{-2.85}$. The cryptanalyst can use this fact to deduce two bytes of information about the key ($K[0] + K[1]$ and $K[2]$) with probability $2^{-10.85}$ instead of the trivial 2^{-16}, and thus reduce the effective key length in exhaustive search by about five bits.

Mantin and Shamir describe in [MS01] a major bias in the distribution of RC4's second output word: it is zero with twice the expected probability of $1/N$, and thus RC4 outputs can be distinguished from random strings by analyzing about 2^8 words of output produced by unrelated and unknown keys. In addition, this bias makes it possible to deduce the second plaintext word in

many broadcast applications (such as groupware or email) in which the same message is encrypted multiple times under different keys. A similar bias occurs in the first pair of output words, which are both zero with probability of $3/N^2$ instead of the expected $1/N^2$. The existence of the second word distinguisher and its improved performance (compared to previously known distinguishers which require 2^{22} times more data) demonstrates the difference between the PRGA and the KSA mechanisms of RC4.

Mantin in [Man01] and Mironov in [Mir02] analyze the distribution of KSA outputs. Both of them calculate, for every permutation value, the distribution of its location in the KSA final permutation and show that the probability of the value a to end up in location b in S at the end of the KSA process is

$$P[S^*[b] = a] = \begin{cases} p(q^a + q^{\bar{b}} - q^{a+\bar{b}}) & a \leq b \\ p(q^a + q^{\bar{b}}) & a > b \end{cases}$$

where S^* is the KSA output permutation, $\bar{x} \stackrel{def}{=} N - 1 - x$, $p = 1/N$ and $q = 1 - p$.

In particular, the probability that value 1 reaches location 0 is 37% *more* than the expected probability of $1/N$, the probability that value 255 reaches location 0 is 26% *less* than the expected probability, and the probability that the KSA output permutation is $\langle 1, 2, \ldots, 255, 0 \rangle$ is about 2^{70} times larger than the expected probability. Mantin also shows how this analysis can be used to connect states which are N *PRGA* (rather than KSA) rounds away. In addition, he analyzes the KSA execution on short keys and shows more biases, caused by the cyclic usage of short keys.

Fluhrer, Mantin and Shamir present in [FMS01] another weakness of the KSA. They identify large classes of weak keys, in which a small part of the secret key determines a large number of bits in the permutation S produced by the KSA. In addition, the PRGA translates these patterns in its initial S into patterns in the prefix of the output stream, and thus RC4 has the undesirable property that for these weak keys its initial outputs are disproportionally affected by a small number of key bits. The length of these weak keys is divisible by some non-trivial power of two, i.e., $\ell = 2^q m$ for some $q > 0$. When $RC4_n$ uses such a weak key of ℓ words, fixing $\Theta(n + q\ell)$ bits of K (as a particular pattern) determines $\Theta(qN)$ bits of the initial permutation with probability of one half and determines various prefixes of the output stream with various probabilities (depending on their length). In addition, this weakness implies that RC4 has low sampling resistance (as defined in [BSW00]), and thus it is more vulnerable to time/memory/data tradeoff attacks.

3 802.11b and Wired Equivalent Privacy

The IEEE 802.11 wireless network standard (which is also known as Wi-Fi) [Com99] is an extremely popular way to connect multiple PC's at home or in the office without using cables. Since such a wireless network is very vulnerable to eavesdropping and misuse, the standard specifies the Wired Equivalent Privacy (WEP) encryption protocol which can be used to make the security of wireless communication comparable to that of a wired network. Since its publication, the security of WEP was subjected to a continuous debate between its designers at the Wireless Ethernet Compatibility Alliance (WECA) and the cryptographic community.

WEP uses a 40-bit secret key (which was the largest easily exportable key when WEP was designed), shared between all the users and the network access point. For every packet, the sender

chooses a new 24 bit Initialization Vector (IV), and the 64-bit RC4 key is the concatenation of the chosen IV (occurring first) and the shared key (occurring last). Such an IV-based mode of operation is commonly used in stream ciphers in order to generate different PRGA outputs from the same long term key, and the frequent resetting of the PRGA is designed to overcome the unreliable nature of the Wireless LAN environment. In addition, WEP computes a CRC-32 checksum over the data payload, appends it to the plaintext and encrypts both parts in order to verify the integrity of the data.

Several vulnerabilities of this WEP protocol were discovered and analyzed in the last few years. These vulnerabilities are only briefly described in this note (see [Arb] for detailed overview of the published results). The simplest weakness is the small size of the secret key and the IV: A 40-bit key can be recovered by an exhaustive search in less than one day. In addition, the limited size (2^{24}) of the IV space implies that IVs (and thus key-streams) are reused during the encryption of different packets. Consequently, this mode can be attacked by constructing a dictionary of all the 2^{24} IVs along with their corresponding key streams. In addition, WEP defines no easy mechanism for changing the shared key, and thus the key is usually changed only infrequently, increasing the attacker's chance to construct this dictionary. Other known attacks exploit the poor authentication to spoof messages. However, these are secondary attacks which do not allow the attacker to derive the key with an attack which is faster than exhaustive search. The first "real" attack which makes it possible to derive an arbitrarily long key in time which grows only linearly (rather than exponentially) with its length in the weakest attack model of known plaintext and IV (rather than the stronger related key or chosen IV attacks) was developed in [FMS01], and is outlined in the next section.

4 The WEP Attack

In order to carry out the attack, the cryptanalyst needs the first output word of a large number of RC4 streams along with the IV that was used to generate each one of them. Since in WEP the IVs are transmitted in the clear and the first message word in most packets is a known constant (see [SIR01]), these requirements are automatically satisfied.

Let us begin by defining our notation. To indicate an RC4 state after using keywords $[0, \ldots, r]$ (i.e., after $r+1$ rounds) during KSA we use the subscript r in S_r, i_r, j_r and t_r (thus the KSA output S^* is also denoted S_{N-1}). In addition, we define several functions over permutations, related to applying the first PRGA round to the given permutation. $I(S)$, $J(S)$, $T(S)$ and $Z(S)$ are the state indices, the output index and the output word of the first PRGA round, given S as an input. Thus $I(S) \stackrel{def}{=} 1$, $J(S) \stackrel{def}{=} S[1]$, $T(S) \stackrel{def}{=} S[1]+S[S[1]]$ and $Z(S)$ is usually $S[T(S)]$ (unless $T(S)$ is $I(S)$ or $J(S)$). K represents the ℓ-word RC4 key, which is a concatenation of the initialization vector IV (whose length is denoted by ℓ'') and the secret key SK (whose length is denoted by ℓ').

4.1 The Basic Idea

The basic step of the attack (which we call the sub-attack) assumes knowledge of some prefix of the RC4 keywords, and derives the next keyword (which we sometimes denote as the target keyword). The attack starts with the known IV as a basis, and repeatedly applies the sub-attack in order to recover all the keywords in the secret key SK.

The sub-attack: Let us assume that the first x words of the KSA key are known. This makes it possible to simulate the first x rounds of the KSA and compute the permutation S_{x-1} and the indices i_{x-1} and j_{x-1} at that point. The next value of i is also known ($i_x = x$) but the next value of j (j_x) depends on the unknown target keyword $K[x]$ (since $j_x = j_{x-1} + S_{x-1}[x] + K[x]$) and thus each of the values j_x and $K[x]$ can be easily derived from the other. Now let us concentrate on the value $S_x[x]$. This value is swapped into this position in round x and thus it was in position j_x just before this round. Consequently, given $S_x[x]$, we can compute which value was in position j_x in the known permutation S_{x-1}, and by inverting this permutation, we can recover j_x itself.

Thus the target keyword depends on j_x, which depends on $S_x[x]$. The main idea of the sub-attack is to identify and isolate cases where the first output word (which is assumed to be known) is correlated with $S_x[x]$ and thus with the target keyword. By analyzing such cases, the sub-attack uses the known output word to guess the target keyword with a significant success probability. This attack is probabilistic rather than deterministic, but it can be repeated with multiple IVs in order to make the statistical information about the target keyword more robust.

4.2 The Details

We apply this attack by concentrating on cases in which the first output byte z has a relatively high probability to be equal to $S_x[x]$. In order to create such a situation, we have to guarantee that:

- z is defined with high probability only by known permutation values.
- The most likely value of z is equal to $S_x[x]$.

Because i scans through the permutation, the values that occur to the right of x at round x participate in an exchange (and are almost always changed) at some later round. However, values that occur to the left of x (including position x itself) have a reasonably high probability to remain untouched until the end of the KSA, since the index j chooses $N - x$ random values among the N possible values in the remaining rounds, and thus it misses any particular value with probability which is at least the constant e^{-1} (and which gets even higher as x increases).

The first output word z is determined by only three permutation values (those in locations $I(S)$, $J(S)$ and $T(S)$), as illustrated in Figure 3. In

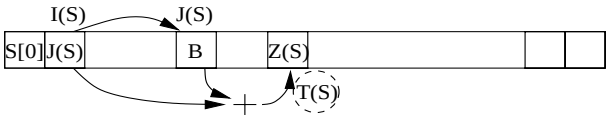

Figure 3: RC4 first output word

order to satisfy the first requirement, we want $I(S_x)$ and $J(S_x)$ to be in positions $0, \ldots, x$ of S_x. In order to satisfy the second requirement, we want to make sure that the index $T(S_x)$ of the output word is exactly x. In other words, we are interested in cases in which $1, S_x[1] < x$ and $S_x[1] + S_x[S_x[1]] = x$. When this happens we say that the state of RC4 is in an *x-resolved* condition, and the first output word has a high probability to be $S_x[x]$. In addition, since the state after x rounds is completely determined by the first (known) x keywords, we can define for every x a class of keys, which we denote as *x-good* keys[2], which are guaranteed to lead to a x-resolved state after x KSA rounds (on class of such keys is described in Figure 4).

[2]RC4 keys, i.e., a concatenation of IVs with shared WEP keys.

When an x-good key occurs, the values in positions $1, S[1]$ and $S[1] + S[S[1]]$ have a relatively high probability to remain untouched during rounds $x+1, \ldots, N-1$. The probability that three locations will not be pointed to by a pseudo random index during the remaining $N-1-x$ rounds is better than $((1-\frac{1}{N})^N)^3 \approx e^{-3} \approx 5\%$. Thus, when an x-good key occurs, the output word is equal to $S_x[x]$ with probability of at least 5%. As specified above, the target keyword can then be guessed from the first output word z through the relations $K[x] = j_x - j_{x-1} - S_{x-1}[x] = S_{x-1}^{-1}[S_x[x]] - j_{x-1} - S_{x-1}[x] = S_{x-1}^{-1}[z] - j_{x-1} - S_{x-1}[x]$. This guess is correct with probability better than 5%, which is twelve times higher than the 1/256 probability of guessing an incorrect value for $n = 8$. We can thus carry out a voting process, in which each x-good key votes for some target keyword candidate, and we can identify the correct keyword with probability better than 0.5 by examining just 60 x-good keys[3].

Hence, if we know some x-prefix of the RC4 key, we can derive the next keyword by analyzing only 60 pairs of an x-good key and the first output word it produces. By starting with the known IV part of the key and repeatedly using this technique, we can recover the keywords one by one until the whole key is extracted. The time complexity of the attack is negligible, and the main difficulty in applying this attack is the collection of sufficiently many good keys. We thus have to estimate how many (IV, z) pairs are needed in order to completely restore the ℓ'-word secret key when it is preceeded by an ℓ''-word non-secret IV.

The full analysis of this issue is described in [FMS01]. As shown in that paper, the fraction of x-good keys for $n = 8$ increases from $2^{-14.42}$ for $x = 3$ to $2^{-10.44}$ for $x = 16$, (the probability increases with x since for large x there are more acceptable values for $J(S_x)$). The number of pairs required for the sub-attack is $60 \cdot \frac{1}{fraction}$, which varies between $2^{20.32}$ (for $x = 3$) and $2^{16.35}$ (for $x = 16$). Notice that an x_1-good key is not necessarily x_2-good, and thus the same collection of (IV, z) pairs can be used to find the 60 good pairs for all the possible values of x. Consequently, the total number of pairs which are needed by the complete attack is the maximum, rather than the sum, of the corresponding numbers in the sub-attacks. In particular, we can attack WEP (for $n = 8$) with an arbitrarily long key and randomly chosen 3-word IVs by collecting only $2^{20.32}$ (IV, z) pairs. If we modify WEP by using 16-word IVs and 16-word shared key (as used in the "improved" version which is unofficially known as WEP2), the data complexity of the attack actually drops to $2^{16.35}$.

Some implementations of WEP use counters (instead of randomizers) in order to generate the IVs. In order to analyze the security of such variants, we introduce several classes of IVs that are likely to lead to x-good keys. Consider for example all the IVs of the form $(x, 255, V, \ldots)$.

Figure 4 describes the state evolution when an RC4 key with such a prefix is used.

Since $J(S_2) = S_2[1] = 0 < x$ and $T(S_2) = S_2[1] + S_2[S_2[1]] = x$, the state after 3 KSA rounds is x-resolved, and thus most keys that start with this pattern are x-good.

If the IVs are generated by a multibyte counter in little endian order (and hence the first byte of the IV increments the fastest), then the attacker can search for IVs of the form $(x, 255, V)$ for $\ell'' \leq x \leq \ell - 1$ and different Vs. By looking only at such IVs the attacker gets a single good pair for every

[3] An important observation is that x-good keys can be correctly identified by tracking the first x KSA rounds. If we had to carry out the voting process with random rather than good keys, the probability distribution of votes would have been much flatter.

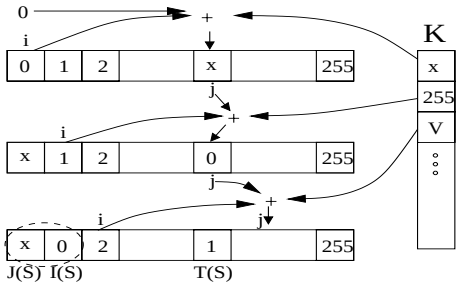

Figure 4: First KSA rounds for keys of the structure $\langle x, 255, V, \ldots \rangle$. The 1 in position x is likely to be swapped with $S_{x-1}[j_x]$ during round x.

```
// This procedure recovers the key
// Parameters meaning:
//   Pairs: a set of ⟨IV, z⟩ pairs,
//   ℓ'', ℓ': the sizes of the IV and the secret key SK
Attack(Pairs, ℓ'', ℓ')
  kp ← NULL // an (x − ℓ'')-word known prefix of SK
  For x ← ℓ''..ℓ − 1 // The size of the known prefix
    c ← SubAttack(x, Pairs, ℓ', kp)
    kp ← kp ∘ c

  Return kp
```

```
// This procedure recovers the next keyword
// Parameters meaning: as in the main procedure
SubAttack(x, Pairs, ℓ'', kp)
  For Every ⟨IV, z⟩ ∈ Pairs
    K ← IV ∘ prefix // Known prefix of the RC4 key
    [i_{x−1}, j_{x−1}, S_{x−1}] ← KSA(K, x)
    // Apply x KSA rounds on the key K (|K| ≥ x)
    If (1 < x, S_{x−1}[1] < x, S_{x−1}[1] + S_{x−1}[S_{x−1}[1]] = x)
      Vote for S^{−1}_{x−1}[z] − j_{x−1} − S_{x−1}[x]

  Return Most popular candidate
```

Figure 5: The pseudo-code of the attack.

x after going through all the combinations of the first two words (2^{16} pairs), and he gets 60 good pairs after searching through at most 2^{22} pairs.

If the IVs are generated by a multibyte counter in big endian order (and hence the last byte of the IV increments the fastest), then the amount of data that is required for the attack depends on ℓ''. For $\ell'' = 3$ it is $2^{16}\ell$ (e.g., 2^{20} for $\ell = 16$). However, this approach works well only for short IVs. Other cases can be obtained by exploiting other 3-good keys (such as the $(V, N-V, N-2)$ and the $(V, N-V+1, N-1)$ sets) or by guessing the first unknown keywords and then looking for IVs that lead to a resolved permutation under the guessed assignment[4].

5 Summary

Shortly after the publication of a preliminary version of [FMS01], Stubblefield, Ioannidis and Rubin ([SIR01]) implemented the attack and successfully derived a 128 bit WEP key by passively observing a real network during a single evening. Several optimization techniques can reduce the required amount of data to the number of packets sent on a fully loaded network in less than 15 minutes. The implications of this attack were described by Ron Rivest in a tech-note that was published several weeks later at the RSA web site ([Riv01]):

"Those who are using the RC4-based WEP or WEP2 protocols to provide confidentiality of their 802.11b communications should consider these protocols to be "broken", and to plan remedial actions as necessary to mitigate the attendant risks. Actions to be considered should include using encryption at higher protocol layers and upgrading to improved 802.11b standards when these become available".

One proposal to modify WEP in response to this attack is TKIP. In this proposal, the key supplied to the KSA is constructed by concatenating the 3 byte IV with a hash of the secret key and the IV. Furthermore, the IV is limited to values that tend not to form x-resolved conditions for small x, and the secret key is updated at least once every 10,000 packets.

[4] This approach is further described in [FMS01].

While we believe that the core idea of RC4 remains sound, we are worried about the lack of robustness of the KSA process which makes the encryption algorithm completely insecure in several natural modes of operation. We recommend either a complete overhaul of the KSA part of RC4, or at least a modification which discards the first N bytes (or more) produced by the PRGA.

References

[Arb] W. A. Arbaugh. 802.11 security vulnerabilities. http://www.cs.umd.edu/~waa/wireless.html.

[BSW00] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In *FSE'2000*, 2000.

[Com99] LAN/MAN Standard Comitee. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999 edition. IEEE standard 802.11, 1999.

[Fin94] H. Finney. an RC4 cycle that can't happen, September 1994.

[FM00] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In *FSE'2000*, 2000.

[FMS01] S. R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *SAC'2001*, 2001.

[Gol97] J. D. Golić. Linear statistical weakness of alleged RC4 key-stream generator. In *EUROCRYPT'97*, 1997.

[Jen98] R. J. Jenkins. ISAAC and RC4. Technical report, 1998. http://burtleburtle.net/bob/rand/isaac.html.

[KMP+98] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) RC4. In *ASIACRYPT'98*, 1998.

[Man01] I. Mantin. The security of the stream cipher RC4. Master's thesis, The Weizmann Institue of Science, October 2001. http://www.wisdom.weizmann.ac.il/~itsik/RC4/thesis.html.

[Mir02] I. Mironov. (Not so) random shuffles of RC4. In *Crypto'02*, 2002.

[MS01] I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *FSE'2001*, 2001.

[MT98] S. Mister and S. E. Tavares. Cryptanalysis of RC4-like ciphers. In *SAC'98*, 1998.

[Riv01] R. Rivest. RSA security response to weaknesses in key scheduling algorithm of RC4. Tech note, RSA Data Security, Inc, October 2001.

[Roo95] A. Roos. A class of weak keys in the RC4 stream cipher. sci.crypt posting, September 1995.

[SIR01] A. Stubblefield, J. Ioannidis, and A. D. Rubin. Using the fluhrer, mantin and shamir attack to break WEP. Technical Report TD-4ZCPZZ, AT&T Labs, August 2001.

ABOUT RSA LABORATORIES

An academic environment within a commercial organization, RSA Laboratories is the research center of RSA Security Inc., the company founded by the inventors of the RSA public-key cryptosystem. Through its research program, standards development, and educational activities, RSA Laboratories provides state-of-the-art expertise in cryptography and security technology for the benefit of RSA Security and its customers.

Please see www.rsasecurity.com/rsalabs for more information.

NEWSLETTER AVAILABILITY AND CONTACT INFORMATION

CryptoBytes is a free publication and all issues, both current and previous, are available at www.rsasecurity.com/rsalabs/cryptobytes. While print copies may occasionally be distributed, publication is primarily electronic.

For more information, please contact:

cryptobytes-editor@rsasecurity.com.

RSA SECURITY®

RSA Security Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie