Quantifying Effect of Network Latency and Clock Drift on Time-driven Key Sequencing

Geoffrey G. Xie* Cynthia Irvine† Tim Levin†
Department of Computer Science
Naval Postgraduate School, Monterey, CA 93943
{xie,irvine,levin}@cs.nps.navy.mil

Abstract

Time-driven Key Sequencing (TKS) is a key management technique that synchronizes the session key used by a set of communicating principals based on time of day. This relatively low cost method of session key synchronization has been used in specialized distributed systems with low-end communicating devices where sessions are sparse and each session spans a short time period comprising a small number of messages.

In this paper, we describe how TKS may be useful in several scenarios involving high speed computer networks. More importantly, we present a performance model of TKS and conduct a detailed analysis to determine the impact of clock drift and network latency on the required key refresh rate. We give the exact conditions for determining the range of adequate key refresh rates, and demonstrate that the derived conditions are sufficient to ensure that data are both protected and deliverable. Interestingly, these conditions may be used to obtain a key refresh rate that can tolerate a maximum amount of clock drift after other parameters in the system are fixed.

1. Introduction

Time-driven Key Sequencing (TKS) is a key management technique that synchronizes the session key used by a set of communicating devices based on time of day. The sequence of session keys is either dynamically generated from a root key or preloaded into each device. This relatively low cost method of session key establishment has been used in specialized distributed systems where sessions are sparse and each session spans a short time period comprising a small number of message exchanges. Typically, in these systems some communicating devices, such as low-end smart cards, have too little processing capacity to support a time-durable crypto-algorithm which features a static key for all sessions spanning the entire system lifetime. Nor can these devices afford an elaborate key management scheme in which a new session key is regularly distributed to the devices through open communication channels. This is because the key transfers must be encrypted and the associated tasks would impose a significant amount of processing overhead at each device.

Compared to most other key synchronization techniques, TKS has a clear advantage for incurring little processing overhead. TKS also has several limitations. First, it requires some form of system wide clock synchronization to bound the maximum clock drift within a tolerable range. Additional overhead may be incurred in other parts of the system to address the security concerns associated with dynamically generating or preloading a sequence of keys. Fortunately, most of this overhead occurs very infrequently on the order of days or months. Second, a system may use TKS to synchronize encryption keys *only if* each piece of data being encrypted has a lifespan that is shorter than the maximum lifetime[1] of the key used for the encryption.

A widely used TKS application is the SecureID authentication device produced by RSA Security, Inc. These devices are used to enhance the security of remote dial-up accesses to corporate servers. Most of them are essentially low-end

*supported in part by DARPA under the Next Generation Internet Program (AO# 417) and by a grant from NSF (ANI-0114014).

†supported in part by DARPA Quorum Program and the Defense Information Systems Agency.

[1]The maximum lifetime of a key equals the minimum amount of time it takes to compromise the key using brute-force.

smart cards with keypad and a small display area. Each SecureID card is preloaded with user-specific root keys that it shares with the target server. When initiating a dial-in session, a card user first enters a user-specific pin code into the card and retrieves a one-time passcode, which is a function of the user's root key and the current value of the card's local clock. The function is implemented by a proprietary hash algorithm. The user then provides this passcode as an additional authenticator to the server along with the pin. To verify the passcode the server first derives its own version of the passcode from the current value of its own clock and the root key that it has stored for that user. The server then compares the two codes and grants the dial-in session only if they match.

Although TKS-based security applications like SecureID are widely deployed, performance studies of these systems are very limited. The few articles that we are able to find in the literature focus exclusively on the potential security risks of RSA SecureID based systems [8, 9]. Little analysis has been done to quantify the adverse effects of clock drift and network latency on the performance of systems implementing TKS. These systems assume the time scale of clock drift and network latency is several magnitude smaller than the rekeying interval and is thus negligible. Because of this assumption, the key refresh rate is often set in an ad hoc fashion and may not be optimal.

The main contribution of this paper is a set of exact conditions for determining the range of adequate key refresh rates based on clock drift and network latency parameters. We prove that the derived conditions are sufficient for the corresponding TKS system to ensure that data are both protected and deliverable. Interestingly, these conditions may be used to obtain a key refresh rate that can tolerate a maximum amount of clock drift after other parameters in the system are fixed. The result also indicates that in general TKS can tolerate large clock drift, on the order of minutes.

We also describe how TKS may be useful in several scenarios involving high speed computer networks. TKS may prove advantageous for some applications even though the processing capacity of individual components appears not to be a limitation for implementing time-durable cryptographic algorithms. In these scenarios, it may be desirable to have a high rate of rekeying that is on the order of minutes and thus approaches the time scale of clock error and network latency. That is where determining a safe rekeying rate becomes a very important concern.

The remainder of this paper is organized as follows. In Section 2 we describe a couple of networking scenarios in which TKS may be favorable. A system model is presented in Section 3 to provide a high-level context for TKS. The performance measures used in this paper are introduced too. In Section 4, a canonical TKS implementation is described, along with intuitions for the negative effects of clock drift and network latency. Section 5 follows with a formal analysis of TKS and two theorems that permit us to bound the rekeying frequency based upon network latency, clock drift, and key durability. In Section 6, a method for choosing the optimal rekeying frequency based on system requirements is presented. Related work is discussed in Section 7.

2. Potential TKS Usage Scenarios

In this section, two potential scenarios in which time-driven key sequencing may prove advantageous are described. The first requires a high speed packet authentication service to counter Denial of Service (DoS) attacks, while the second requires message confidentiality for data that has a short lifespan.

We define "cryptographic mechanism" to include specific selections for all variable parameters, including, where applicable, the transformation algorithm (including the number of internal "rounds" or permutations), blocksize, and the length of the key. The security or "cryptographic strength" of such a mechanism can be taken to be a measure of its resistance to attack, and may be a function of some or all of these parameters. Similarly, the specific parameters of a cryptographic mechanism will determine its level of resource use, or overhead. Processor utilization is the most critical overhead. In general the stronger is a cryptographic mechanism, the more processing overhead it incurs. For example, the performance of the Advanced Encryption Standard (AES) algorithm (Rijndael)[4], is functionally dependent on the size of the key used. Both setup and encryption take longer as the key length increases.

Scenario 1. The Internet routing protocols such as IS-IS, OSPF and BGP frequently flood the network with link state or path update packets to keep routing tables synchronized and up to date. In each of these update cycles, a router must process a large number of control packets sent by numerous other routers. Worse, these packets typically arrive in bunches, creating "packet storms" that could overload the receiving routers. The problem will be exacerbated when various proposals for achieving subsecond route convergences are implemented [1, 2].

An adversary will explore this problem and launch DoS attacks against these protocols rather easily as follows. The

adversary identifies those routers already busy processing control packets and then aggravates their predicament by sending a large number of faked or duplicated control packets to them. Such attacks will delay routing table convergence and disrupt network services.

An effective measure to counter these DoS attacks is deploying to each router interface a packet filter to form a *first-line defense*. Malicious packets are dropped early and rapidly before they enter the more processing-intensive stages. The main design objective for the filter is "minimal processing overhead", as long as security is not compromised. Although the existing security extensions of the Internet routing protocols can detect malicious packets, they do not meet the requirement of minimum processing overhead. Their main objective is "adequate security". TKS, on the other hand, provides an ideal basis for developing the packet filter. By increasing the rekeying frequency, the system may use a "weak-per-key" but lightweight cryptographic mechanism like keyed-MD5 to achieve the same filtering accuracy (i.e., 100% drop rate of malicious packets) as a strong but costly mechanism based on public key cryptography.

We have developed such a filter as part of our effort to build a link-layer high speed packet authentication protocol [10]. Running as part of the NetBSD kernel on an antiquated Pentium 200Mhz PC box, the filter was still able to achieve a filtering rate of 75 Mbps.

Scenario 2. This scenario centers upon confidentiality. Use of rapidly changing keys would be useful in situations where time-critical data are distributed in encrypted form to a group of subscribing customers. As an example, consider time-critical analyses of highly volatile financial markets. Here subscribers are provided with timely multimedia analyses to support informed investment decisions in a fast-paced market. The analyses are of considerable value for a few hours, but are available to the public after a fixed period and provide no added value to subscribers thereafter. Subscribers receive continuous updates and analyses. They may join and leave the subscriber pool periodically, for example monthly, thus we are not concerned with rapid rekeying in order to include or exclude particular subscribers or former subscribers respectively. We are, however concerned with attackers who wish to access the valuable sensitive information for free. Because attackers will be able use both known ciphertext and known plaintext, it is necessary to change the keys while subscribers are actively using the service. Yet, the fact that the information is no longer sensitive after a few hours means that the cryptographic mechanism used needs only to be sufficiently strong to protect the information for these relatively small intervals. Again, security related processing overhead may be reduced by using the right, not necessarily the strongest, cryptographic mechanism.

In both scenarios, it may be beneficial to use a high rekeying frequency. But because of non-negligible clock drift and network latency in real systems, there is an upper bound on the maximum rekeying frequency. On the other hand, the minimum rekeying frequency is directly related to the maximum key lifetime. The remainder of the paper is dedicated to these topics.

3. A System Model of TKS

While TKS is applicable to different types of network communication protocols, for ease of presentation we assume a packet-based (e.g., IP) network. The following definitions applys to the System Model used to discuss TKS in this paper.

- System. A set of communicating nodes, one or more of which comprise a logical (viz, potentially distributed) key distribution center (KDC). Nodes wish to communicate securely through the shared use of a cryptographic key table. The nodes can be of various types ranging from powerful workstations and gateways to lighter-weight appliances. Therefore, the TKS processing requirement at the node should be minimized. One may also interpret a node as a logical entity such as a user account or even an application activated on a physical machine.

- Maximum Key Lifetime, T. Each cryptographic key used in the $System$ is subject to various attacks as soon as the first packet secured with the key enters the public network. The amount of time that the key can remain concealed while under attack is defined to be the *maximum lifetime* of the key.

- Key-Table. An ordered set of session keys. The same table must be available at each node of the system. These keys Key table distribution can be accomplished by several means. One method is to transmit "seed" keys (e.g., see [3]) which are used by the nodes as part of an algorithm for local generation of the complete key table. Another means is to distribute the literal keys that make up the key table. Literal keys, to the extent that they are generated "randomly," will exhibit more inter-key independence, as algorithmically

generated keys will be related to each other, however remotely, via the algorithm.

For the purposes of this paper, the key-table has an infinite size; this is understood as reflecting an implementation mechanism which replenishes all nodes with additional sequence of keys before the expiration of the last key in the current sequence, thus infinitely extending the length of the table.

- Time-driven Key Sequencing, TKS. In a system utilizing shared logical key tables, TKS is a method for synchronizing the transition between keys without utilizing explicit node-to-node handshaking.

3.1. Threat model

The threat model assumed in this paper is one of insecure communication channels between System nodes, such that cryptographically protected data packets might be intercepted and then subjected to attacks on the key space. We do *not* consider with attacks aimed at directly compromising the underlying hardware and/or software of a node. We assume that the prefetched keys (or their seeds, in the case of algorithmically-generated keys) are stored and distributed securely.

3.2. Performance measures

We are primarily concerned with two performance measures. The first one is a Boolean indicator of *security*. Specifically, the described System is said to be secure if keys are not used beyond their maximum lifetime at any node, per the TKS protocol, even in the context of an insecure communication channel. A formal definition is given in Definition 1 of Section 5.

If a key is used beyond its maximum lifetime, the system may be vulnerable to a brute-force attack as follows. An intruder who intercepts packets from the message stream of a trusted node can use cryptanalysis techniques or a search of the key space to discover the key, and then hijack the message stream. The receiver node will not be able to detect this intrusion just by inspecting the messages since the intruder is able to modify all packet fields (including any timestamp or sequence number) after learning the key. Because of such attacks, it is also not advisable to synchronize session keys with a limited maximum lifetime by adding a key index field to packets.

The second performance measure is about *data delivery*. It describes the efficacy of the TKS system in delivering useful data while trying to meet the security requirement. The specific metric we use is called *maximum tolerable network latency*, denoted by D and a performance target that can be set *a priori*. A TKS system is said to support maximum tolerable network latency of D seconds if a node in the system will never drop a valid packet — one that comes from another node in the system and is not tempered with in transit — unless the packet is delayed more than D seconds by the network. A formal data deliverability condition is defined in Lemma 1 of Section 5.

4. Implementing TKS with Dynamic Key Windows

In this section, the general behavior of a TKS system is described using a reference implementation in which each node stores active session keys in special variables named *dynamic key windows*. The implementation is referred to as TKS-DW. We hope the ensuing discussions will also provide some intuitive explanations for the adverse effects of clock drift and network latency.

In TKS-DW, each node maintains two dynamic key windows. The first is the *Send-Key Window*, which is of size 1. The node uses the session key in this window to cryptographically transform outgoing packets. A special term (s) is designated for the time when a key is first put into the send-key window. The duration of a key staying in the send-key window is fixed and is called the *Key Window Period*, W. An obvious constraint needed on W is that it be less than or equal to the cryptographic lifetime of one key. At the end of the current key window period, a *Send-Key Window Transition* occurs and a new key is put into the send-key window.

We will motivate the need for a *Receive-Key Window* and then give its precise definition. The effect of clock drift is considered first and for the moment network latency is assumed to be negligible.

In a perfect network where the local clocks of all of the nodes are exactly synchronized, all nodes would make send-key window transitions at precisely the same time. All senders and receivers would see the same key in their respective send-key windows.

Perfect clock synchronization does not exist in real networks. If a receiver's clock is slow with respect to that of a sender, then a current key of the sender will appear to be either a current key or a future key to the receiver. Similarly,

a current key of a sender will be either a previous key or the current key of a receiver with a relatively fast clock. What is of interest to note is that between any particular sender-receiver pair, a key will either be a previous/current key or a current/next key at the receiver, not both. This means that, for a particular sender-receiver pair, a key will be active for a duration equal to a variable number, n, of Key Window Periods, i.e., nW.

Because the clock of a given receiver may be slow with respect to some senders and fast with respect to others, the receiver must maintain a receive-key window that encompasses both previous, current, and future send-keys. Thus the *Receive-Key Window* defines a set of one or more keys that a node uses to cryptographically transform (e.g., validate) incoming packets. The window also has a finite period of W. In other words, one key in the window is replaced every W time.

The receive-key window has the following generic positions:

- x previous send-keys
- 1 current send-key
- y next send-keys

We wish to minimize the number of keys that must be inspected upon receipt of a packet. In what follows, the size of the receive-key window is assumed to be 3 (i.e., $x = y = 1$). This assumption places limitations on the allowed clock drift, which will be explained below.

As a key window period expires, a *Receive-Window Transition* takes place and the receive-key window "slides" down the key table in the following manner. The key in each position of the receive key window is replaced by its "next" neighbor, such that in the subsequent receive-key window, (1) the current key has become the newest previous key, (2) the oldest next key has become the current key, (3) a new key has entered the table (as the newest next key) and (4) the oldest previous key has left the table.

Figure 1 illustrates how key windows move with time at a node. s denotes the time when k_i, the key at the ith row of the table, becomes the current sending key. Recall that W is the *Key Window Period*. In time interval (a), k_i is the current sending key and it is in the receive key window along with k_{i-1} and k_{i+1}. At the end of the interval, i.e., exactly[2] $s + W$, the node slides the key windows down one row of the table. The node has now entered window interval

[2]For ease of presentation, we assume that the node uses k_i as the sending key at $s + W$ and switches to k_{i+1} at $(s + W)^+$.

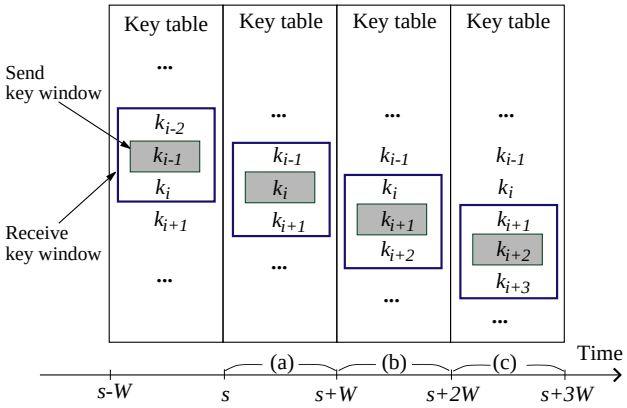

Figure 1. Key windows of a node at different time intervals

(b) and k_i is the previous key. The node then slides the key windows one more row at time $s + 2W$. At this point the node has entered window interval (c) and k_i is no longer in the receive-key window. Observe that with perfect clock synchronization, a key stays in receive-key window for two consecutive window periods after becoming the send key and subject to attack. In that case, the key lifetime must be greater than $2W$ to ensure security.

Suppose the clock drift of each node is bounded with respect to a standard time source by a value e. The maximum clock time that two nodes can drift apart with respect to each other is $2e$. Thus the receive-key window period must be extended by $2e$ to ensure data delivery in the worst-case, which occurs for a (fast sender, slow receiver) pair. Consequently, the lower bound on key lifetime must be increased from $2W$ to $2W + 2e$.

What are the consequences of network latency? The time that it takes a packet p to get from source to destination is called the *network latency of* p and is denoted $d(p)$. This means that the key window period W must be at least as large as $d(p)$ to have a chance to not drop p mistakenly. When also taking into consideration a maximum of clock drift $2e$ between the sender and receiver, W must be at least $D + 2e$ to not drop packets whose network latency can be as large as D.

5. Exact conditions for using TKS

In the last section, we presented intuition that there are exact limits on the key window period W (and thus the

rekeying frequency) with respect to the maximum key lifetime, clock drift, and network latency. In this section, we formally establish that these limits are sufficient to ensure that data is both protected and deliverable under TKS-DW with the receive-key window size set to three keys.[3] The security objective of TKS-DW is formally defined below.

Definition 1 *Assume that the maximum lifetime of each key is T seconds measured by standard time. The protocol is not vulnerable to brute-force attacks if it never uses a key beyond the maximum key lifetime; that is, the protocol is not vulnerable if the following holds for any key k that it uses,*

$$t_l(k) - t_f(k) \leq T, \tag{1}$$

where $t_f(k)$ is the standard time when the first packet secured with k enters the public network, and $t_l(k)$ the last standard time that k is in a receive-key window at any node.

Equation (1) tells us that the protocol is secure when the keys are used within the time bounds of their defined lifetimes, given standardized (i.e., perfectly synchronized) clocks at all nodes. However, in reality, the clocks at local nodes may deviate from standard time. Theorem 1 presents an condition under which we can guarantee that the protocol is secure using local, non-standard, clocks.

Assumption 1 *Each node has a local clock. A mechanism is in place to synchronize these clocks with the standard time so that at any time the absolute difference between any local clock and the standard time is upper bounded by e seconds. That is, at any standard time t and for any node n in the TKS system,*

$$\mid c_n(t) - t \mid \leq e, \tag{2}$$

where c_n denotes the local clock of n, and $c_n(t)$ the reading on that clock at standard time t.

Theorem 1 (Condition for Security)
Consider a TKS-DW system. Let W be the key window period. Assume that the maximum lifetime of each key is T seconds. If the following security condition holds

$$W \leq \frac{T}{2} - e, \tag{3}$$

then for every key k used by the system,

$$t_l(k) - t_f(k) \leq T. \tag{4}$$

[3]The results can be extended to arbitrarily large receive-key window sizes.

The proof of Theorem 1 is straightforward based on the intuition given in the end of previous section. (See [11].) Combining the results of Definition 1 and Theorem 1, it is clear that a TKS-DW system is secure if it meets the condition specified by equation (3). Thus, we have derived a condition for TKS-DW to ensure security.

Next, we turn attention to the adverse effect of network latency on packet delivery. The network latency of a packet is measured as the difference between the standard time when the packet leaves the sending node and the standard time when the packet arrives at the receiving node. Given a specific maximum tolerable network latency target D, Theorem 2 establishes the importance of choosing W based on D to ensure that data are deliverable.

Lemma 1 *Consider an arbitrary packet p secured by a TKS sender. Let k be the key that secures the packet. $a(p)$ denotes the standard time when the packet arrives to the receiving node, and $t_l^p(k)$ the last standard time that k is in the receive-key window of the receiving node of p. If the packet has not been tempered with, and*

$$t_l^p(k) - 3W \leq a(p) \leq t_l^p(k), \tag{5}$$

then the packet will be accepted by the receiving node.

Theorem 2 (Condition for Data Deliverability)
If W meets the following data deliverability condition

$$W \geq D + 2e, \tag{6}$$

then for any packet p whose network latency (denoted by $d(p)$) does not exceed D,

$$t_l^p(k) - 3W \leq a(p) \leq t_l^p(k). \tag{7}$$

Lemma 1 follows directly from the protocol specification of TKS-DW and the fact that $(t_l^p(k) - 3W, \leq t_l^p(k)]$ is the receive-key period of k at the receiving node. The proof of Theorem is straightforward based on the intuition given in the end of previous section. (See [11].) Combining the results of Lemma 1 and Theorem 2, it is clear that if the condition specified by (6) holds, then a valid packet, i.e., one that is secured by a TKS sender and is not tempered with in transit, will be accepted by the receiver as long as its network latency does not exceed D. Thus, we have determined a condition on TKS-DW to ensure deliverability of data.

6. Practical Uses of the Conditions

We see several practical uses of the theoretical results presented in the previous section. They are described below.

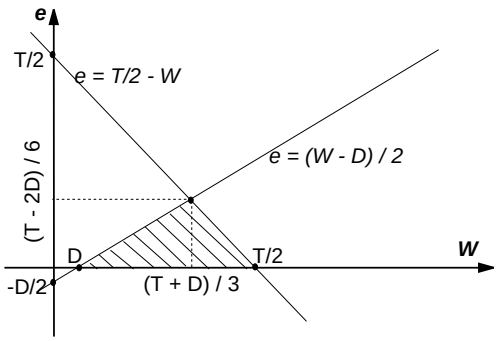

Figure 2. Relationship between e and W

6.1. Choosing W to maximize tolerance of clock drift

Assume that the key lifetime T and the maximum network latency tolerance D have already been determined based on system needs. Next we will discuss the relationship between the key window period W and the maximum tolerable clock drift e. We will show that there exists a W value that maximizes allowable e. That is, *given T and D one can choose a particular key window period to maximize the system's clock drift tolerance.* Although a time synchronization protocol such as the Internet Network Time Protocol (NTP) can be used to reduce e to tens of milliseconds, it is still desirable to maximize the system's clock drift tolerance because the NTP could be under attack itself or malfunction, in which case e would be much higher than normal.

From the security and deliverability conditions of Theorem 1 and 2, the following upper bound on e can also be established

$$e \leq \min\{\frac{W-D}{2}, \frac{T}{2} - W\} \tag{8}$$

Now we have a range of e to choose from based on W. If the value of W is set too close to $T/2$ or D, then from equation (8) e is required to be very small, which may be infeasible.

The W value that maximizes e can be derived based on equation (8) as follows. Figure 2 shows the e-W relationship embedded in equation (8). There is a feasible region, the shaded triangle of the figure, of (e, W) combinations that meet both security and deliverability conditions. Within that region, when $W = \frac{T+D}{3}$, e is maximized at $\frac{T-2D}{6}$.

As an example, let $D = 30$ seconds, which is much larger than the normal network latency in the Internet. Furthermore, let $T = 1800$ seconds, which is a conservative estimate for current cryptograhic systems. If clock drift is the principal concern, one should choose a key window period that is close to $\frac{(1800+30)}{3} = 610$ seconds so that a maximum clock drift of $\frac{(1800-2\times30)}{6} = 290$ seconds can be tolerated.

6.2. Determining D and e

When selecting a value for the maximum network latency tolerance D one should consider how the application reacts to packet delays. For example, it would be ok to set D to a very small value (i.e., subsecond) if the system is to just carry live voice traffic. Overly delayed packets are useless to this type of applications. Similarly for typical Internet Web applications, the value of D can be in the range of seconds.

It would be prudent to make a conservative estimate on maximum clock drift e. Several existing time synchronization protocols can bound the clock drift within a second. Global Positioning System (GPS) satellite based clocks may reduce e further to the order of 1-2 microseconds. However, these performance numbers are achieved in normal operating situations. The time synchronization protocols or GPS satellites may malfunction or be under attack. Therefore, it is important to select an e value that is sufficiently large to account for extra clock errors caused by unusual system conditions.

Our analysis suggests that the security of a TKS system will not be compromised by an overestimated e or an improperly set D value. This is because the security condition is independent of D and a larger e value causes keys to be refreshed more frequently. On the other hand, the data delivery performance of the system may be significantly impacted by these values. Other factors such as rekeying overhead need to be considered too in determining the right D and e values for a system. (See sections below.)

6.3. Choosing W based on D, e, and T

After T, D and e are determined, a range of W values may be feasible. Based on the security and deliverability conditions, specifically equations (3) and (6), the window period W can be selected from the following range

$$D + 2e \leq W \leq \frac{T}{2} - e \tag{9}$$

For example, let $D = 20$ seconds, $e = 20$ seconds, and $T = 1800$ seconds. Then W can be any value between 60 and 880 seconds.

The value of W determines how often a new key is needed. When there is a range of values to select from, W should be as large as possible. The main advantage of this approach is that it minimizes the overhead of key generation and/or distribution. The trade-off is that large W requires long-lasting keys and possibly more processing overhead associated with the transform.

6.4. Choosing the right cryptographic mechanism

For well-designed cryptographic systems, the per packet transform overhead typically increases with the maximum key lifetime (T). To reduce the overhead, it is desirable to use a cryptographic mechanism whose maximum key lifetime is just right for the task at hand. After D and e are determined for the system, a lower bound on W can be obtained based on the deliverability condition, i.e., equation (6). Once W is set, a lower bound on T can be determined from the security condition, i.e., equation (3). Finally, an appropriate cryptographic algorithm may be selected based on this lower bound on key lifetime.

For example, consider designing the packet filter described in the usage scenario 1. let $D = 20$ seconds and $e = 10$ seconds. From equation (6), the lower bound on W is $(20 + 2 \times 10) = 40$ seconds. Let's assume that W can be set to a minimum of 120 seconds due to key generation and distribution overhead and other concerns. Then from equation (3), T has a lower bound of $(2 \times 100 + 2 \times 10) = 220$ seconds. This lower bound of T can be used to select a (transform, key-length) pair with just the right strength.

7. Related Work

Several security risks of a particular RSA SecureID implementation and recommended fixes are reported in [9]. All of the problems seem to have something to do with how the SecureID system is engineered and operated, not the TKS concept itself. A cryptanalysis of the RSA SecureID's supposedly proprietary hash algorithm is presented in [8].

The IPSec protocol [7] provides a framework for managing encryption and authentication, and their associated policies, at the network (IP) level. The default automated key management protocol for IPSec is referred to as Internet Key Exchange (IKE) [5]. Key exchange is based on the use of the Diffie Hellman algorithm, which is relatively computationally intensive. By default, the lifetime of a IPSec session key must at least as long as the duration of the target session. One may avoid using a long duration key by partitioning the traffic of a long session into several sub-flows. However, each of these flows requires additional set-up overhead[4] [6]. For usage scenarios described herein, TKS would be ideal for replacing IKE or other expensive key exchange algorithms (viz, instead of using IKE to change keys for sub-flows of a long session).

8. Concluding Remarks

There is a limit to how much efficiency can be gained from more frequent rekeying. At some point, the per-key overhead of changing keys can overtake the resource advantage of using more efficient cryptographic mechanisms. Further work is needed to quantify the characteristics of this effect.

References

[1] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards millisecond igp convergence. Internet Draft *draft-alaettinoglu-isis-convergence-00.txt*, Nov. 2000.

[2] A. Basu and J. G. Riecke. Stability issues in ospf routing. In *Proceedings of ACM SIGCOMM 2001*, pages 225–236, San Diego, CA, Aug. 2001.

[3] B. Briscoe. MARKS: Zero side effect multicast key management using arbitrarily revealed key sequences. Presented at 46th IETF meeting, Dec. 1999.

[4] J. Daemen and V. Rijmen. Available from *csrc.nist.gov/encryption/aes/round2/AESAlgs/Rijndael/Rijndael.pdf*, June 1998. NIST AES Proposal.

[5] D. Harkins and D. Carrel. The interenet key exchange (IKE). RFC 2409, Nov. 1998.

[6] A. Kara. Protecting privacy in remote-patient monitoring. *IEEE Computer*, pages 24–27, May 2001.

[7] S. Kent and R. Atkinson. Security architecture for the Internet Protocol. RFC 2401, Nov. 1998.

[8] Mudge and Kingpin. Initial cryptanalysis of the RSA SecurID algorithm. On-line White Paper, Jan. 2001. Available from *www.atstake.com/research/reports/initial_securid_analysis.pdf*.

[9] PeiterZ@silence.secnet.com. Weakness in SecurID. On-line White Paper. Available from *www.tux.org/pub/security/secnet/papers/secureid.pdf*.

[10] G. G. Xie, C. Irvine, and C. Colwell. LLPA: A protocol for high speed packet authentication. Technical Report NPS-CS-99-003, Department of Computer Science, Naval Postgraduate School, Feb. 1999.

[11] G. G. Xie, C. Irvine, and T. Levin. Conditions for time-driven key sequencing. Technical Report NPS-CS-00-001, Department of Computer Science, Naval Postgraduate School, Aug. 2000. Revised, July 2001. Available from *www.cs.nps.navy.mil/people/faculty/xie/papers.*

[4]The idea of key prefetching for long duration flows is briefly mentioned for IPSec. However, no details for its realization are given.