Enhancing Privacy with Shared Pseudo Random Sequences

Jari Arkko, Pekka Nikander, and Mats Näslund

Ericsson Research

{jari.arkko, pekka.nikander, mats.naslund}@ericsson.com

February 17, 2005

Abstract

Protecting users' privacy is essential for turning networks and services into trustworthy friends. Many privacy enhancing techniques, such as anonymous e-cash and mix-nets, have been proposed to make users more comfortable in their network usage. These techniques, in turn, usually rely on very basic security mechanisms, e.g., confidentiality protection, for their realization. But these mechanisms are also used for other security related reasons.

In this paper, we make some new observations on how security can degrade privacy. For example, using security as a component of an advanced privacy enhancing technique may not have the effect we expect; i.e., too careless application of security may defeat the assumed privacy gains. In particular, introducing new identifiers may make it *easier* to track users. This effect is especially harmful to mobile users. Even in cases when privacy is not the main driver for the use of security, we believe that identifiers require special attention in some circumstances.

We propose a mechanism, which we call to allow the communicating parties to continuously change the identifiers they use, without any signalling and without adverse affects on realibility or security.

1 Introduction

In communication systems where the user terminals and/or the users are mobile, preventing tracking of users and equipment is important for privacy reasons. The main challenge in preventing tracking is to avoid the use of long-term or easy-to-correlate protocol information that constitutes explicit "identifiers" or otherwise allows users to be identified. Even if the identifiers cannot be tied to a physical entity, they may make possible to follow the same entity as it moves from one place to another (where the "place" may be geographical, i.e. physical, or logical, e.g. a network address).

Some telecommunications mechanisms take this into account already now, and can use frequently and/or randomly changing identifiers. For example, in GSM, the so-called TIMSI, Temporary IMSI (International Mobile Subscriber Identifier), is used to hide the true IMSI. However, in general, such techniques are not useful unless they are enforced *throughout the protocol stack*. For instance, while Wireless LAN authentication mechanisms can employ pseudonyms, see [5, 2], or even completely hide the authentication exchange from others [6], this is of limited value as long as fixed link layer identifiers (e.g. MAC addresses) are used at a lower layer.

The problem exists in many forms. A particularly visible example is the transmission of cleartext, human-readable user identities such as Network Access Identifiers (NAIs) [1]. Similar problems appear for the transmission of stable but "meaningless" identifiers, such as IP addresses [10]. A less known problem is that even data that is completely independent of any real "identifier" can be used to track users. For instance, an IPsec SPI [8] can reveal that a node in one place is the same node as a node that appears later in another location, if the SPI value has not changed even though the IP addresses are no longer the same. For example, with a 32-bit SPI, the chance is about 1 in 4 billion that it is not the same user if the SPIs are the same. This is particularly problematic for Internet Key Exchange (IKE) Security Parameter Indices (SPIs), as there is no possibility for efficiently renegotiating IKE SPIs without revealing the previous SPIs in the process. For IPsec SPIs this is less of a problem, as the SPIs can be re-negotiated within the protection of the IKE SA, hence hiding the change from outsiders. Nonetheless, the problem remains that privacy enhancing measures can sometimes be defeated by unexpected factors.

The same problem arises in certain authentication mechanisms. For authentication purposes, a popular techniques is the use of public key cryptography. For efficiency reasons, symmetric cryptography based counterparts are gaining popularity in the form of so-called hash chains. A quite well-known problem with public keys is that the key, even if not tied to an identity, leaves "traces" of the user, since anybody can verify authenticity using the public key. However, one easily forgets that also a hash chain is easily linkable in the forward direction by applying the hash.

Even data that changes for every packet can be used to track users. For instance, TCP or IPsec sequence numbers may in some cases be sufficient for the identification of equipment even if no other stable identifiers are present. As long as the sequence number space is sufficiently large and nodes distributed along to a sufficient degree, a node that presents a sequence number N in one place and $N + 1$ (or something close to it) in another place shortly thereafter is likely to be the same node.

1.1 Related work

Hiding identifiers and other communications inside a protected tunnel or tunnels is used in protocols such as TLS or IPsec. The drawback of this solution is that often other identifiers still remain visible outside the "tunnel".

Another approach is the use of “pseudonyms”. This approach is used in GSM [11], among others. In this technique, an identifier is used for login to a service, and the service returns an encrypted token that the client can decrypt and use as the identifier for logging into the service the next time. A drawback of this scheme is that the new pseudonym has to be returned, which adds to the amount of signalling necessary. In any case, this solution may not be possible in all situations. For instance, the protection of sequence numbers in this manner would be possible in TCP as there are ACKs, but would be hard in IPsec because there may not be traffic in the return direction before a new packet needs to be sent. In any case, waiting for the new pseudonym before a second packet can be sent is inefficient.

Removing sequence numbers (and thereby linkability) may be considered as an option. However, with present art this is not a universally viable, as it creates a sender/receiver synchronisation problem, at least when used with unreliable data transport mechanisms such as IP.

For public keys and hash chains, an available method to improve privacy is to frequently generate new public keys/hash chains. However, this is computationally quite expensive.

Recently, [9] proposes some mechanisms for address location privacy in MIPv6.

As mentioned in the abstract, some very sophisticated techniques for sender-receiver untraceability have been proposed, e.g. Chaum’s mix-nets [3]. However, such mechanisms are far too cumbersome to use for “everyday” communication, and relies on the help of “mixing nodes”.

2 A practical example

Let us consider an example where Alice and Bob are communicating over the Internet, using IPsec to protect their connection[1]. Initially, Alice learns Bob’s IP address from the DNS, and she initiates IKE by sending an UDP packet to Bob. Bob learns her current IP address from the received UDP packet. As a result of running IKE, Alice and Bob agree upon an IKE SPI, a pair of Encapsulated Security Payload (ESP) SPIs, and ESP sequence numbers[2]. With the IP addresses, SPIs, and sequences numbers both Alice and Bob and any outsiders can easily keep track of the ongoing conversation. Furthermore, the IP addresses reveal the approximate location of the parties.

Now consider a situation where Alice moves to another location. As a consequence, her IP address is changed. This requires some action from her in order to inform Bob about the new address. One option is to run IKE again, and create new Security Assocatiations (SAs) and corresponding SPIs. Unfortunately, this is pretty costly, may involve user interaction with token cards, and

[1]For brevity, we ignore the link layer and physical aspects, but the principles presented in this paper could be equally applied there.

[2]IKE protects the public keys against passive attackers; we don’t need to consider them here.

with frequent movements would cause disruptions to the connection. Another option would be to use an underlying mobility service such as Mobile IP [4]. All traffic now flows through Alice's home agent, hiding her new IP address but allowing outsiders to easily link her communications from different places. If an advanced form of mobility, route optimization, is used then all packets carry a new identifier, Alice's home address. This can then be used to link different packets, even if sent from different places.

A third option is the use of IKE extensions (such as NAT traversal or MOBIKE) that allow hosts to change their IP address without requiring IKE to reauthentication. Presumably, at the same time she could negotiate new ESP SPIs. Unfortunately, the IKE SPI would still reveal her identity, allowing our eavesdropper to continue tracking the connection and to learn the new IP address (and approximate location).

To summarise the current situation, Alice has basically two options: either she can gain some privacy at a computational or routing cost, by re-running IKE or using Mobile IP without route optimisation, respectively, or alternatively she can suffer a privacy loss with the gain of not needing to re-run IKE or suffer sub-optimal routing.

3 Identification via Pseudo Random Sequences

To overcome the privacy problems caused by identifiers and other easily trackable protocol values, we propose a method where the constant identifiers or easily predicatable protocol values are replaced with values drawn from a number of shared, secretly agreed pseudo-random sequences. That is, as the parties initiate a shared communication context they agree on a number of pseudo-random sequences, constituting a sequence of sets of identifiers, in addition to the usual session keys and other protocol state.

Whenever the sending party wants to avoid identity tracking, it draws the next set of identifiers in the sequence and uses those identifier values in the outgoing messages instead of the previous values. This practise works best whenever there would be a natural rupture in the tracking sequence anyway, e.g., when the sending party has just changed its location or has been inactive for a while. This could even happen for every packet. For full privacy, it is crucial that the sending party replaces the identifiers (and other trackable values) at all protocol layers at the same time, to prevent linkage across layers.

3.1 Example: Adding Privacy to the MOBIKE Protocol

Let us reconsider the example above. With the proposed MOBIKE protocol in place, the only value that revealed Alice's identity was the IKE SPI, s. With our new mechanism in place, instead of agreeing on just a single fixed SPI value, s, Alice and Bob would agree on a *sequence* of SPI values, s^0, s^1, s^2, etc, denoted as $\{s\}^*$. The intial IKE negotiation would use the first value from this sequence, s^0, in the place of the original fixed value. Now, when Alice has moved and received a new IP address, she would use the next value in the sequence, s^1. As this value belongs to the sequence that Bob has agreed with Alice, he would still recognize the message being sent by Alice[3]. However, any outsiders that do not have access to the sequence cannot link the SPI values together.

To generate the sequence, Alice and Bob can simply use a keyed pseudo random number generator with a shared key K derived from the keying material created during the initial IKE negotiation. Two basic approaches are discussed below:

Pseudo-random Function: Alice and Bob use a keyed pseudo-random function (not necessarily invertible), f, and the jth identifier ID^j is simply $f(K, j)$, or more generally $f(K, j||\text{IDtype})$. In practice, a cryptographic hash such as SHA-256 can be used as f.

Pseudo-random Permutation: Alice and Bob use a keyed pseudo-random permutation (i.e. an invertible function), π, and the jth identifier ID^j is simply $\pi(K, j)$, or, $\pi(K, j||\text{baseID})$. This enables authorized receiver to reconstruct the baseID, which could be advantageous in some cases. The permutation π could be instantiated using a block cipher with apropriate block size, or, using a stream cipher $ID^j = s(k, j)$ xor baseID.

Using the method requires no protocol changes; it is enough that Alice and Bob mutually agree on the pseudo random number generator and how to derive the shared key from the keying material. Such an agreement could be provided out-of-band. However, if Internet-wide deployment is desired, a simple extension can be defined to IKE in order to negotiate the parameters; see Appendix 5.

3.2 Avoiding cross-layer correlation

In order to be effective, the method should be applied to all visible identifiers and other linkable protocol fields. Furthermore, moving from one set of values to a next set must be properly synchronized through the stack so that no value can be used to link the value sets together. For example, consider a complete protocol stack that includes the link, internetworking, transport, and application layers protocols L, I, T and A, with linkable identifiers and values ID_L^A, ID_I^A, ID_I^B , ID_T^A, ID_T^B, SN_T^A, SN_T^B, VAL_{A1}, VAL_{A2}, corresponding to the Alice's link layer address, Alice's and Bob's IP addresses, the transport layer port and sequence numbers, and some application layer values. Alice would then define pseudo random number sequences for each of these values. The first sequence, $\{ID_L^A\}^*$, need not be communicated to Bob as it is only used locally. However, it needs to be changed in synchrony with the other values to prevent correlation. The second and third sequences, the IP addresses, may not be fully known at the time the connection is initiated due to movements of the hosts[4]. All the

[3]Message integrity and originality is still protected by the same session key.

[4]However, in IPv6 it would be possible for the two nodes to agree on the interface identifier parts of their IP addresses. This resembles the idea of cryptographically generated addresses [**?**], but instead of address ownership these interface identifiers would be used for privacy purposes, and be dynamic.

other sequences, end-to-end in nature, Alice and Bob need to agree upon.

3.3 Sending packets

In general, our method does not require any changes to existing protocols, other than providing a mechanisms for the parties to agree on using the method. All modifications are local to the parties.

The changes at the sending end are fairly simple. When the sending party decides to move from one set of identifiers to the next set, it simply does so. That is, when Alice wants to break any on-going linkage, she moves to the next element all of the above mentioned sequences $\{ID_L^A\}^* \dots \{VAL_{A2}\}^*$, and sends the next packet using the new values. Typically, such a change would be triggered by a natural change of some value, e.g., the IP address, but in principle it can also be initiated at any time. Apart from any local means needed to associate any lower layer addresses together (such as ARP or IPv6 Neighbor Discovery), no other action is needed by Alice as the sender.

3.4 Matching received packets

At the receiving end the situation is more complex. In general, the receiver must be prepared not only receive packets using the current identifier values, but in case of reordering or packet loss, also some past and future values.

In an unmodified protocol, the receiver would accept packets that are identified with a set of identifiers $ID_L \dots ID_A$, corresponding to different protocol layers, and a serial number that fits in the window $SN_T^A \dots SN_T^A + \delta$. In our method, the identifiers would follow the sequences $\{ID_L\}^* \dots \{ID_A\}^*$ and, depending on the use of the serial numbers, the serial numbers would not be communicated at all or would be periodically changed by drawing new values from a corresponding sequence. If the receiver currently expects the jth set of identifiers, then it would accept any identifier sets within the range $f(K, j), f(K, j + 1), \dots f(K, j + k)$, and update j accordingly. Alternatively, with the pseudo-random permutation approach, the receiver would retrieve the sender's j from $\pi^{-1}(K, j)$.

In a typical hash-table based packet demultiplexing system, the added processing cost is neglible. In the case of receiving a packet with the expected jth set of identifiers, the code basically works just like today. In the case of identifiers that the hosts have full control over, the situation is simple: instead of accepting packets only at the jth set of identifiers, the host also accepts packets on the next k identifier sets.

In some cases accepting packets simultaneously on several identifier sets may lead to collisions. Typically, the packet hash table lists identifiers for all active remote peers. In an unmodified system, the identifiers are allocated in a manner that makes sure that there are no collisions between any identifier sets in use. When identifier sequences are used, there is the possibility that the ith identifier set for one peer may completely or partially collide with the jth set of another peer; in section 3.5 we analyse this in more detail for TCP.

Generalising, the receiver needs to allocate some additional memory for matching received packets, and possibly need to be prepared to resolve some collisions. For most modern computers, such an increase in memory and CPU use is neglible. The actual cost of collision resolution depends on the protocol, but appears to be relatively simple in the typical cases.

3.5 Example: Collisions in TCP

Considering the Transmission Control Protcol (TCP), the receiver is normally prepared to accept packets that are identified with the identifier quadruple $\langle ID_I^A, ID_I^B, ID_T^A, ID_T^B \rangle$, where ID_I^A is the sender's IP address, ID_I^B is the receiver's IP address, ID_T^A is the source port, and ID_T^B is the destination port. Furthermore, the packet must match the TCP sequence number window $SN_T^A \dots SN_T^A + \delta$. In the current stacks, TCP acknowledgements outside of the expected window are simply discarded but the packet is otherwise processed.

To be prepared to receive packets using identifier sets $j+1 \dots j+k$, in addition to the jth expected set, the situation becomes somewhat more complex. In the IPv4 world, an additional source of complexity is the inability to know beforehand what the next IP address will be. As a consequence, the receiver must be prepared to accept packets that are identified with *a*ny source IP address, the current local IP address, and the TCP ports matching any of the identifier sets $j+1 \dots j+k$. In other words, in addition to the expected jth identifier set, it must be prepared to accept packets that are identified by the quadruple template $\langle any, ID_I^B, \{ID_T^A\}^{j+1 \dots j+k}, \{ID_T^B\}^{j+1 \dots j+k} \rangle$. The existing TCP code can be reused to find these potential "shadow" transmission control block (tcb) candidates for any non-matched packet, at the cost of at most a few hundred machine instructions.

The sequence and acknowledgement number checks become slightly more complex. In TCP, the sequence and acknowledgement numbers do not count packets but bytes. Consequently, the window sizes are typically so large that we cannot use the pseudo-random sequences directly for them. Instead, a plausible way seems to keep updating them just as today as long as the jth identifier set is used, and to add the next value to them as the host moves to $j+k$th identifier set.

In more practical terms, for each candidate "shadow" tcb

$$\langle any, ID_I^B, \{ID_T^A\}^{j+i}, \{ID_T^B\}^{j+i} \rangle$$

the corresponding acceptable sequence number window is

$$(SN_T^A + \{SN_T^A\}^{j+i}) \dots (SN_T^A + \{SN_T^A\}^{j+i}) + \delta_A$$

where SN_T^A is the current TCP sequence number as maintained by TCP, $\{SN_T^A\}^{j+i}$ is the $j+i$th element from the generated pseudo-random sequence, and δ_A is the receive window size. Similarily, the acceptable range of acknowledgements numbers is

$$(SN_T^B + \{SN_T^B\}^{j+i}) \ldots (SN_T^B + \{SN_T^B\}^{j+i}) + \delta_B$$

where δ_B is the number of sent but unacknowledged bytes.

When receiving a new packet, the TCP stack would first look for a perfect match. If it finds one, there is little to worry as the tcb is fully qualified by the sender's IP address. However, if the packet does not match any existing tcbs, the stack next needs to look for candidate "shadow" tcbs[5]. Because the source IP address does not qualify the "shadow" tcbs, it is possible that a packet matches more than one. In that case we have a collision; the identifiers alone are insufficient for determining which of the remote peers have moved to use a next identifier set.

To resolve collisions, and to make sure that a candidate "shadow" tcb is really the right one, the receiving host can next check the sequence and acknowledgement windows. If the sequence and acknowledgement numbers in the packet fall within the updated sequence and acknowledgement windows, the right session for the packet is likely to be found. However, as the TCP window sizes can be fairly large, it is possible that the collision cannot be resolved even with the help of sequence numbers.

If there was a collision, the receiving host may want to verify that it has found the right "shadow" tcb. To do so, the receiver can send, using the new identifier set, a TCP packet that carries no data and acknowledges zero new bytes. The packet triggers retransmission, thereby confirming that the session is indeed the correct one. Such practise does not appear any more vulnerable than the current TCP. This optional verification requires the exchange of two packets.

To quantify the situation, we should estimate the probability of two "shadow" tcbs to collide. In TCP, the port numbers are 16-bit values and the sequence and acknowledgement numbers 32-bit values. A typical receive window, δ_A, is in the order of 64 kilobytes or less. For the output window δ_B, we conservatively estimate that its size is equal to the receive window size; usually it is less. Consequently, the probability of a randomly generated packet matching with a given "shadow" (tcb) is (at most)

$$\frac{\delta_A \cdot \delta_B}{|ID_T^A| \cdot |ID_T^B| \cdot |SN_T^A| \cdot |SN_T^B|} = \frac{2^{16} \cdot 2^{16}}{2^{16} \cdot 2^{16} \cdot 2^{32} \cdot 2^{32}} = 2^{-64} \approx 10^{-19}.$$

The probability of a collision between two existing TCP connections depends on the number of simultaneous active TCP connections. Let us consider a host that maintains N active TCP connections, holds state for the sequence values $j \cdots j + k$, and initiates TCP sequence numbers randomly. The probability of having a "shadow" tcb collision between any given two TCP connections (assuming $\delta_A = \delta_b = \delta$) is

$$\frac{\delta^2}{|ID_T^A| \cdot |ID_T^B| \cdot |SN_T^A| \cdot |SN_T^B|}.$$

[5] In the current stack a packet that does not match any existing connection is dropped.

Consequently, the probability that there will be no collisions with any of the $(N-1)k$ incorrect shadows is (at least)

$$\left(1 - \frac{\delta^2}{|ID_T^A| \cdot |ID_T^B| \cdot |SN_T^A| \cdot |SN_T^B|}\right)^{(N-1)k}.$$

As our analysis shows, for most protocols the probability of a packet being falsely accepted or there being a collision between two existing connections is relatively low. In the cases we have analysed, it is trivial to compensate for the collisions by reusing existing protocol mechanisms. For example, if there is a collision between two integrity protected sessions, attempting to verify the message authentication code with both of the possible session keys will determine the right session.

3.6 Network Impacts

The proposed method is end-to-end in nature, and in general does not impact other nodes, as long as packets can be freely sent to their destinations. However, it is fair to note that changing identifiers such as addresses or port numbers can be problematic from the point of view of firewalls, NATs, and network access control tools such as 802.1X, all of which may keep state related to the identifiers. In addition, the efficiency of bridge learning protocols may be impacted by frequent change of MAC addresses.

4 Protecting other parameters

4.1 Public key traceability

In the introduction, we mentioned problems with the traces left by public keys, which we have not really delt with so far. As mentioned, we do not want to frequently generate completely new keys, using expensive number theoretic operations. The main use of public keys in security protocols are for key exchange, and here there are some well-known hybrids between asymmetric and symmetric cryptography that we can use. Specifically, one could use Diffie-Hellman key exchange, authenticated by the shared symmetric key and a MIC, rather than using, say RSA signatures. Since the DH values g^x, g^y are random for each protocol execution, there is no useful information extractable from these and the corresponding MIC values.

4.2 Hash chains

Hash chains may also need to be made unlinkable. This is easily achieved by, instead of using $h_j = H(h_{j-1})$, taking a keyed variant $h_j = H(K||h_{j-1})$. The chain is still easily forwards verifiable, but only by parties sharing K. Note that here, it is important that H is a one-way function, i.e. using a pseudo-random permutation $h_j = \pi(K, h_{j-1})$, would defeat security as the chain then becomes reversible.

4.3 Mobility Parameters

Mobile IP home addresses could be pseudo-random sequences instead of current stable, real IP addresses.

5 Summary

In this paper we have outlined a generic method where fixed or easily predictable identifiers and protocol field values are replaced with values drawn from a set of mutually agreed pseudo-random number sequences. The resulting sequence of protocol packets can be easily processed by the receiving party with modest requirements for additional memory and processing, while being completely untraceable by any outsider observers.

References

[1] Aboba, B. and M. Beadles. The Network Access Identifier. RFC 2486, IETF, January 1999.

[2] Arkko, J. and H. Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). Internet Draft draft-arkko-pppext-eap-aka-15.txt (Work In Progress), IETF, December 2004.

[3] Chaum, D. The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability. J. of Cryptology **1**, 65–75, 1988.

[4] Johnson, D., Perkins, C. and J. Arkko. Mobility Support in IPv6. RFC 3775, IETF, June 2004.

[5] Haverinen, H. and Salowey, J. Extensible Authentication Protocol Method for GSM Subscriber Identity Modules (EAP-SIM). Internet Draft draft-haverinen-pppext-eap-sim-16.txt (Work In Progress), IETF, December 2004.

[6] Josefsson, S., Palekar, A., Simon, D. and G. Zorn. Protected EAP Protocol (PEAP). Internet Draft draft-josefsson-pppext-eap-tls-eap-07.txt (Work In Progress), IETF, October 2003.

[7] Kaufman, C. (Ed.) Internet Key Exchange (IKEv2) Protocol. Internet Draft draft-ietf-ipsec-ikev2-14.txt (Work In Progress), IETF, May 2004.

[8] Kent, S. and R. Atkinson. Security Architecture for the Internet Protocol RFC 2401, IETF, November 1998.

[9] Koodli, R., V. Devarapalli, H. Flinck, and C. Perkins. Solutions for IP Address Location Privacy in the presence of IP Mobility. Internet Drafy draft-koodli-mip6-location-privacy-solutions-00.txt (Work in Progress), IETF, Feb 2005.

[10] Narten, T. and R. Draves Privacy Extensions for Stateless Address Autoconfiguration in IPv6 RFC 3041, IETF, January 2001.

[11] European Telecommunications Standards Institute. Digital cellular telecommunication system (Phase 2); Security related network functions. GSM Technical Specification GSM 03.20 (ETS 300 534), August 1997.

Appendix A. IKE extension

To be filled in.