

Mitigating Insider Threats to RSA Key Generation

Adam Young
Cigital Labs
ayoung@cigital.com

Abstract

RSA keys form the cornerstone for numerous security systems. They provide for confidentiality of communications as well as non-repudiability of digital signatures. However, there are several insider attacks against RSA key generation that can have devastating effects when carried out. In this paper we address such attacks by surveying measures that can be taken to mitigate insider attacks against RSA generation.

1 Introduction

The RSA algorithm [20] has gained widespread use in the software industry and it is utilized by more and more people each year. RSA key pairs are the basis for ensuring both the privacy of data in RSA ciphertexts as well as the non-repudiability of digitally signed messages. However, the security of these basic functionalities rests on the honest and correct generation of RSA key pairs.

There are many subtle security issues surrounding the generation and use of RSA keys. For example, there are instances in which a malicious user may deliberately try to generate and certify a *weak* public key. The user can choose the prime p in the RSA public key $n = pq$ such that $p - 1$ is *smooth* (a smooth integer has no large prime divisors). This allows anyone to factor n using Pollard's $p - 1$ factoring algorithm [19].

The notion of generating a weak key may appear counter-intuitive to many readers. Why would anyone ever want to do a thing like that? The reasons for doing it are many.

Perhaps the biggest reason for doing so is to be able to back out of a contract if business begins to turn south. By certifying a “weak” RSA public key, the signer will be in a position down the road to repudiate any and all digital signatures that were created using the corresponding private key. This can be argued on mathematical grounds in a court of law since a weak key is a key that can be factored by anyone. Hence, everyone has the ability to produce signatures using the weak key pair. Weak keys are particularly attractive to a malicious user when the existence of the weakness is not readily apparent, since in this case another user is not likely to produce forgeries under the malicious user's name.

However, to avoid being blamed for deliberately generating a weak key, the signer would have to convince a disinterested third party (such as a judge) that the use of the weak key was not deliberate. This is a challenge since it must be proven that a trustworthy key generation algorithm was used and that it was not tampered with. When RSA keys are generated randomly, the probability that a key is weak is already very small, and the use of *strong primes* reduces the risk even more. Strong primes have certain properties that make the product n hard to factor by specific factoring methods. Such properties include the existence of a large prime factor of $p - 1$ and a large prime factor

of $p + 1$. This is one of the issues in the strong primes debate.

Why else would a user want to certify a weak key? Consider the possibility of political insurgency. A person from country A starts working for the government in country B and acts under cover. The person certifies a weak public key and uses it to store, receive, and transmit (in key exchange protocols) highly sensitive information. This has the potential to severely damage country B. Also, an “innocent” weakness in the key could get the person off the hook if he or she is accused.

Of course, a simpler approach to the problem is for a malicious user to simply publish his or her private key in some inconspicuous fashion. The user would later point out the location of the private key and state that anyone could have obtained it. However, this argument is not likely to hold up in court. One would have to assess the probability that the private key would show up naturally, without the intervention of the key owner, and the chances of this are very small indeed.

In truth weak keys are not likely to be generated, even when the simplest methods for generating RSA keys is used. However, it is possible using simple RSA key generation. So, a malicious user can try to make the case that, e.g., $p - 1$ just happened to be smooth. These issues illustrate the importance of being able to validate RSA keys [21].

Even an honest user may generate and use an easily factorable RSA public key without realizing it. This happens when a malicious insider, such as the programmer that creates the RSA key generation device, inserts a backdoor that lets the insider obtain the user’s private key. The reasons why a programmer would want to insert such a backdoor are obvious. It would permit the programmer to gain illicit access to encrypted information such as e-mails, secure socket connections, and so forth, and would also allow the

programmer to impersonate users (e.g., forging signatures of other users, accessing the accounts of other users, etc.).

The scope of the problem is by no means specific to RSA. Backdoor attacks have been shown to exist in Diffie-Hellman, ElGamal, DSA, elliptic curve cryptosystems, and more. The scope of the problem is not limited to insider attacks either. An outsider is often in a position to insert a backdoor as well. Malicious software such as viruses and worms can insert a backdoor as part of their payload. The scope of this problem is immense, especially considering the fact that a backdoor can often be exploited in a completely covert way. For instance, when the attacker simply reads information but does not modify information it is often difficult to detect that the attack is even occurring.

A tamper-resistant microchip is an ideal medium for planting a backdoor, since by its very nature the backdoor is well-hidden. Even when a key generation algorithm is implemented in software, the program is effectively a “black-box” in the eyes of the average user, since a deep understanding of mathematics as well as the underlying assembly language is necessary to discern the true nature of the program.

2 Proving the Form of $n = pq$

When an RSA key generation device outputs two RSA primes, the key owner can perform rather simple tests on the correctness of the outputs. For instance, the key owner can check that p and q are primes, that they are the correct size, etc. However, for ensuring non-repudiability it is necessary that *other users* be convinced as well that $n = pq$ was generated properly. The purpose is to convince others that n is not “weak” and that there is no backdoor in use. This is a challenging problem since it is often the case that only the key holder is allowed to know p and q .

Certain properties relating to the correctness of RSA key generation can be verified simply by performing computations on n . For instance, it can be publicly verified that: n is not prime, n is not divisible by small primes, and that n is not a perfect power (see [2] for sieving algorithms). Performing these checks is a good measure, since a weak key such as $n = p^2q^2$ will be readily discovered. These verifications help to show that n was properly generated, but they are not sufficient. For example, when a 1024-bit key n is generated these verifications will not detect the case that p is 160 bits in length and q is 864 bits in length.

One way that a black-box key generation implementation can prove these more complex assertions regarding n is to utilize *non-interactive zero-knowledge proof systems*. In a nutshell, a non-interactive zero-knowledge proof system consists of a proof generating algorithm and a corresponding verification algorithm.

The proof generating algorithm proves some type of assertion regarding a problem instance. For instance, the problem instance may be an integer n and the assertion may be that n is the product of two distinct prime numbers. The output of the proof generating algorithm is a data file that is typically anywhere from 40 kilobytes to over a 100 kilobytes in size. The data file has the property that it reveals nothing about the secrets associated with the problem instance (e.g., it does not expose p or q).

The verification algorithm takes this file and the problem instance as input and verifies whether or not the file is valid. A valid file implies that the assertion holds with overwhelming probability. Several such proof systems can be utilized to prove the form of n . These proof systems are the subject of this section.

The following are some well-known zero-knowledge interactive protocols that show various properties of n . All of these protocols can be converted into non-interactive zero-knowledge proof

systems. Peralta and van de Graaf presented a protocol that proves in perfect zero-knowledge that n is a Blum integer [12]. They define the set of Blum integers to be integers of the form $n = p^r q^s$ where $p, q \equiv 3 \pmod{4}$, r and s are odd, and p and q are prime. A zero-knowledge protocol has been given that proves that n is square-free [6]. Recall that an integer n is said to be *square-free* if m^2 does not evenly divide n for any $m > 1$. The protocol utilizes a parameter k that signifies the number of rounds in the protocol. By making k large enough, a cheating prover has a negligible chance of convincing the verifier that n is square-free when in fact it is not. A protocol that proves that n is a Blum integer combined with a protocol that proves that n is square-free proves that n is contained in the subset of Blum integers characterized by $r = s = 1$.

Zero-knowledge protocols have been developed that prove surprisingly complicated facts about n . For instance, a statistical limited-knowledge protocol (that leaks very little information) has been given that proves that n is the product of two primes that are nearly equal in size, assuming that n has already been proven to be the product of two distinct primes [17]. A statistical zero-knowledge protocol has been given that proves that n is the product of two quasi-safe primes [11]. Finally, there is a zero-knowledge protocol that proves that n is the product of two safe primes [7]. This last protocol is asymptotically efficient but would be cumbersome to utilize in practice.

An interesting open question regarding Blum integers is the following. Is there a probabilistic (or deterministic) algorithm for deciding whether or not n is a Blum integer? The existence of such a predicate would eliminate the need for a zero-knowledge proof that n is a Blum integer.

These algorithms and protocols that prove various properties of n are helpful since they prove that there are no obvious weaknesses in the structure of n . However, they do not sufficiently protect against various

forms of abuse. There is a wealth of literature surrounding the abuse of key generation algorithms, digital signature algorithms, and so on. These abuses are the subject of the next section.

3 Cryptographic Abuses of RSA Key Generation

Gus Simmons initiated the investigation of abuses that involve clandestine information leakage within the context of cryptographic algorithms and protocols. The classic problem that demonstrates this type of abuse is known as the *Prisoner's Problem* [22]. In the prisoner's problem, two prisoners are allowed to communicate to each other but are not allowed to send encrypted messages to each other. They are only permitted to exchange public keys and digitally sign their messages. The problem is to devise a way, using the digital signature algorithm in question, for the two prisoners to communicate secretly with each other through digital signatures in such a way that the warden cannot detect or read the subliminal messages. Such a communications channel is called a *subliminal channel*.

Yvo Desmedt noted that a subliminal channel exists in composites, and that use of this channel constitutes an abuse of RSA key generation [8] (see also [16]). One way to implement a subliminal channel in composites is as follows. A subliminal message m_s and a checksum t of m_s are concatenated together (denoted by $m_s || t$). The resulting string is asymmetrically encrypted using the public key of the recipient of m_s . The asymmetric cryptosystem must be probabilistic to ensure that c is pseudorandom (OAEP can be used [4]). Let c be the resulting ciphertext. A random prime p and a random pad RND are chosen. The quotient q and remainder r are then solved for in $c || RND = pq + r$. If q is composite then this process is repeated. When q is prime, the public key is

$n = pq = (c || RND) - r$. At worst a borrow bit will be taken from c , but this can be rectified. The subliminal message m_s is recovered by decrypting c and $c + 1$ and verifying which of the two checksums is correct. Note that the security parameter for c is half of the value of the security parameter for n . This approach is based on kleptography [26, 25].

The subliminal channel in composites can be used to let one prisoner communicate secretly with another (although they have to keep generating new keys to keep communicating this way). Therefore, typical RSA key generation is subject to information leakage abuse by inmates.

RSA key generation is also subject to abuse by malicious insiders. Consider the following rather simple attack. The attacker, who is the programmer that is creating the RSA key generation algorithm, stores a secret seed in the key generation algorithm and the algorithm supplies this seed to pseudorandom number generator. The fact that the seed is chosen uniformly at random and is "secure" leads to a cryptographically secure pseudorandom bit sequence. This sequence is known to the attacker and can be the sole source of randomness for deriving output pairs (p, q) . The attack amounts to replacing the "honest" random sequence that is inherent to a probabilistic Turing machine with a "dishonest" pseudorandom sequence that is completely reconstructable by the insider.

An RSA key pair that is compromised in this way allows the insider to read anything encrypted using the user's public key, and allows the insider to forge any signed document on behalf of the user. This type of attack is a very general one, since it can be applied to *any* probabilistic algorithm, not just cryptographic ones.

Research has been conducted to investigate insider attacks against cryptographic algorithms with the specific goal of making the attacks robust from the attacker's perspective. This type of attack is far more

attractive to an attacker than generating a “weak” key that can be exploited by anyone (see the attack in Section 1). The goal in this type of attack is to plant a backdoor in the key generation algorithm that: (1) generates keys that are indistinguishable from “normal” keys, (2) is robust against reverse-engineering, and (3) generates keys that are cryptographically secure with respect to everyone *except* the attacker. This type of attack gives the attacker an exclusive advantage.

It has been shown how to use the notion of a subliminal channel to mount this type of attack against RSA key generation [26, 25]. The attack makes use of the subliminal channel in composites $n = pq$ where n is a W -bit quantity. The intuition behind the insider attack is as follows. If there were a way to display randomly generated information in the bit representation of $n = pq$ such that: (1) only the insider can access the information, (2) only the insider can detect that the information is there, and (3) the information allows the insider to factor n , then a robust attack against RSA key generation would exist. The fact that the information is randomly generated each time that a key pair is generated provides security going forward with respect to a passive reverse-engineer.

A heuristic version of this attack is as follows. It makes use of a pseudorandom number generator (PRNG) denoted by G . The insider places his or her own public key in the device. This key is used to compute c in the subliminal channel. The device chooses m_s randomly. It then computes the pseudorandom bit sequence $G(m_s)$. The bits in this sequence are considered $W/2$ bits at a time. The first such sequence that is a $W/2$ -bit prime becomes p . If p leads to a prime value for the quotient q in the channel, then $n = (c || RND) - 1$ is output as the user’s public key. The insider obtains this public key from a CA, for instance. The insider then uses his or her own private key to obtain m_s . Given m_s it is then straightforward to recover p and factor n .

The attack is robust against reverse-engineering since only the public key of the insider, not the corresponding private key, is revealed upon inspecting the RSA key generation code. Furthermore, compromised composites are computationally indistinguishable from uncompromised composites under reasonable intractability assumptions, thus assuring that no one will ever know that the attack is being carried out [26, 25].

This type of attack is called a secretly embedded trapdoor with universal protection (SETUP). The attacker’s public key is the secretly embedded trapdoor. The advantage of a SETUP attack over using a fixed pseudorandom bit sequence is that it provides secrecy going forward. That is, even if the key generation device is reverse-engineered and its state is revealed, it will not help the reverse-engineer factor the future (or even past) RSA keys that are produced. This is because the seed m_s is chosen randomly each time that the key generation algorithm is invoked. This is of particular importance in software implementations in which each user obtains the exact same copy of the key generation software.

These types of attacks are by no means unique to RSA key generation. In addition they have been shown to exist in discrete-logarithm based cryptosystems [27]. A SETUP attack has been shown against the Diffie-Hellman key exchange [10] that leaks one of the two Diffie-Hellman exponents. An attack has been shown against the Digital Signature Algorithm that leaks the private key, and other attacks have been shown as well.

4 Early Work on Curbing Abuses of Cryptosystems

The standard approach to mitigating insider abuse in a cryptographic algorithm is to prevent the algorithm from having the luxury of controlling the final ran-

domness that is used to derive the output values. This is enforced by requiring the use of a protocol to perform the computation, as opposed to a stand-alone algorithm.

Gus Simmons introduced the idea of using randomization to destroy subliminal channels [22]. To destroy a particular subliminal channel that was identified, Simmons has the warden generate a random number $x \in R_n$ (the ring of residues modulo n) and modify the message that was being sent from one prisoner to the other prisoner using x . For other early results that use randomization to eliminate subliminal channels, see [23, 9].

To address the problem of key generation abuse, Desmedt investigated *abuse-free* ways of generating key pairs. His protocol is outlined within the context of the prisoner’s problem, but applies to other abuses as well such as the previously mentioned attack that uses a pseudorandom number generator. In this protocol an inmate Alice and a warden jointly generate a key pair such that only Alice knows the private key and such that the warden is convinced that no form of abuse is occurring [8].

A high-level description of this protocol is as follows. Alice commits to a random string r_a and the warden sends Alice a random string r_w . Alice performs the bitwise exclusive-or operation to obtain the random string $r = r_a \oplus r_w$. If r does not satisfy the properties needed to make the key pair (e.g., it does not lead to two RSA primes) then Alice reveals r_a to the warden. If r does lead to a proper public key, then Alice proves this in zero-knowledge. The overhead of this zero-knowledge protocol is substantial in practice since it relies on a very general zero-knowledge interactive protocol construction [13].

5 Oracle-Based RSA Key Generation

In this section a heuristic algorithm is presented that minimizes the capacity of the subliminal channel in RSA composites $n = pq$. The entire approach is not provably secure, although its design utilizes best-practices (it is similar to the process of generating DSA parameters) and it is ideal for use in stand-alone RSA key generation programs since it does not involve a protocol. The key generation algorithm is “oracle-based” since it utilizes a *random oracle*. In a nutshell, a random oracle is a deterministic function that returns a random string in response to a given query. It is an idealized function that is approximated in practice using a cryptographic hash function.

Before continuing, some basic notation needs to be introduced. The Greek symbol ϕ denotes Euler’s totient function. The greatest common divisor function is denoted by gcd . The notation $|A|$ is used to represent the number of bits in the bit string A . For example, when $A = 00101$ it follows that $|A| = 5$. Finally, $\{0, 1\}^\infty$ denotes the set of countably infinite bit strings.

5.1 RSA Key Generation

Recall that in the RSA algorithm, the public key is (n, e) where n is the product of two large randomly chosen primes p and q that satisfy $gcd(e, \phi(pq)) = 1$. The value e is the public exponent, which must be an odd integer greater than 2. The private key is d where d satisfies the equality $ed = 1 \bmod \phi(n)$. As is typically the case in practice, it will be assumed that e is fixed and shared by everyone (e.g., $e = 2^{16} + 1$).

In a typical RSA key generation algorithm, a $W/2$ -bit integer p is chosen randomly and is accepted only if p is prime and $gcd(e, p-1) = 1$. If p is rejected then p is reassigned to be $p+1$, it is tested for primality, and

$\gcd(e, p - 1)$ is computed again. This process repeats until a prime p is found such that $\gcd(e, p - 1) = 1$. This is called *incremental search*. Most standards recommend that p be at least 512 bits in size. This same procedure is used to generate the prime q . There are many variations of this algorithm. For example, before testing for primality p and q often have their 2 most significant bits set to 1. Other methods involve generating each prime such that the prime minus one has a large prime divisor, checking the $p - q$ is large, and so on.

A significant weakness in this standard approach to generating RSA keys is that it exhibits a subliminal channel that is capable of displaying about $W/2$ -bits of explicitly chosen information in the bit representation of pq . When exploited, this channel can display any potentially sensitive information, and hence paves the way for devastating insider attacks.

The algorithm *GenPrivatePrimes1* models a typical method of RSA key generation based on independent generation of primes. This algorithm sets the stage for the more robust key generation algorithm called *GenPrivatePrimes2* that is presented in Subsection 5.2. *GenPrivatePrimes1* outputs two large prime numbers p and q that satisfy $\gcd(e, \phi(pq)) = 1$. The primes also satisfy other constraints as well.

The purpose of *GenPrivatePrimes1* is not to “replace” existing methods of RSA key generation. Rather, it is a hypothetical algorithm that is used to demonstrate the security of *GenPrivatePrimes2* that utilizes a cryptographic hash function. *GenPrivatePrimes2* is more robust than straightforward RSA key generation (e.g., *GenPrivatePrimes1*) since it significantly reduces the number of bits that can be subliminally displayed in the bit representation of composites pq . It is later shown that these two algorithms produce pairs of primes that are drawn from the same set and probability distribution. So, the purpose of *GenPrivatePrimes1* is to show just how “normal”

(as far as RSA key generation goes) the primes are that are output by *GenPrivatePrimes2*.

For simplicity, *GenPrivatePrimes1* only generates $W/2$ -bit primes such that W is evenly divisible by 2. Given (p, q) , it is a simple matter to compute $n = pq$ and $d = e^{-1} \bmod \phi(n)$.

RandomBitString1():

input: none

output: random $W/2$ -bit string

1. generate a random $W/2$ -bit string str
2. output str and halt

GenPrivatePrimes1():

input: none

output: $W/2$ -bit primes p and q such that
 $p \neq q$ and $|pq| = W$

1. while (TRUE) do:
2. $p = \text{RandomBitString1}()$
3. if $p \geq 2^{W/2-1} + 1$ and p is prime then
 goto step 4
4. while (TRUE) do:
5. $q = \text{RandomBitString1}()$
6. if $q \geq 2^{W/2-1} + 1$ and q is prime then
 goto step 7
7. if $|pq| < W$ or $p = q$ then goto step 1
8. if $\gcd(e, \phi(pq)) \neq 1$ then goto step 1
9. set $S = (p, q)$
10. output S , zeroize all values in memory, and halt

The purpose of the seemingly trivial function *RandomBitString1* will be addressed in more detail later on. It is abstracted away since it constitutes a tempting target for the insertion of a subtle backdoor.

Testing for primality can be performed in deterministic polynomial-time [1]. The expected running time of finding p can be found using the *prime number theorem* which was proven independently by Hadamard and De La Vallée Poussin in 1898 (see [15] for a

statement of the theorem). The prime number theorem implies that a random $W/2$ -bit number will be prime with probability about $2/W$. The choice of p in step 2 is independent each time around, so the while loop that selects p corresponds to independent trials (Bernoulli trials) with a fixed failure probability. In m iterations of the while loop, the probability that no acceptable prime p is found is about $(1 - 2/W)^m$. So, this can be used to estimate the chances of finding p . The same analysis applies to finding q .

The reason that the algorithm checks whether or not $|pq|$ is less than W in step 7 is to ensure that n is a W -bit composite. It is standard practice to generate RSA moduli n that are exactly 768 bits in length, 1024 bits in length, and so on. It is constructive to consider how $|pq| < W$ can occur.

Note that a K -bit string can have leading zeros. A K -bit positive integer must have a most significant bit equal to 1. The product of two K -bit positive binary integers with $K > 1$ is either a $2K$ -bit integer or a $(2K - 1)$ -bit integer. To see this, note that 2^{K-1} is the smallest K -bit integer. 2^{K-1} squared is 2^{2K-2} , which is $2K - 1$ bits long. Also, note that $2^K - 1$ is the largest K -bit positive integer. $2^K - 1$ squared is $2^{2K} - 2^{K+1} + 1$, which is a $2K$ -bit integer. So, in this algorithm it is entirely possible that $W/2$ -bit primes p and q will be chosen such that $|pq| < W$ bits.

5.2 The Oracle-Based Key Generation Algorithm

The basic idea behind the oracle-based key generation heuristic is to fill the subliminal channel in composites with a hash value and force the key generation device to reveal a pre-image for this hash. It is similar in flavor to the NIST method for generating DSA keys that was devised, in part, due to the allegations of some researchers that DSA could be using “trapdoor” primes (that might permit signatures to be forged) [18]. To convince skeptics that trapdoor

primes were not in use, a procedure was developed for generating DSA parameters using a one-way hash function, namely, SHA. The DSA parameter generation algorithm is given in Appendix A of [24].

The main idea behind the DSA parameter generation algorithm and the algorithm given in this section is as follows. In this approach, a pre-image to the hash algorithm is generated randomly and is supplied to the hash algorithm. The parameters are derived directly from the output of the hash algorithm. This high level description glosses over various details, since an integer counter is used to speed things up, for instance.

In the algorithm proposed by Smid and Branstad [18], the DSA parameters p and q are output by the parameter generation algorithm along with a pre-image and counter value that can be used to heuristically verify how p and q were generated. This procedure makes it very difficult to choose p and q and then find the corresponding pre-image and counter value that give rise to them. So, assuming that a very small number of pairs (p, q) are amenable for use as a backdoor, it should be difficult to find a pre-image and counter value that leads to one of these pairs under this hashing procedure.

The goal in this section is to devise a lightweight RSA key generation algorithm that is as robust as possible against insider abuse without using a trusted third party during key generation, and without relying on multiple, independently manufactured programs that work together to generate keys. The operating assumption is that key generation is performed in a black-box device. When the output values are verified, the private key holder and the CA can heuristically verify that at most a very small number of bits are being subliminally leaked in $n = pq$. This assures that a high-bandwidth SETUP attack is not being performed.

This key generation algorithm can replace any stand-alone RSA key generation implementation. The

black-box can output other values as well, such as a non-interactive zero-knowledge proof that n is a Blum integer. This provides even more assurance to the CA that n is a properly generated RSA public key.

A little background on the theoretical “tool” known as a *random oracle* is needed [3]. Recall that a random oracle $R(\cdot)$ is a deterministic function from $\{0, 1\}^*$ onto $\{0, 1\}^\infty$ that behaves like a random function. That is, R always returns the same response for a particular input string, and for a given input string the output is drawn uniformly at random from $\{0, 1\}^\infty$ (this sampling is performed once and for all when R is defined).

Let $H(s, i, v)$ denote a function that invokes the oracle and returns the v bits of $R(s)$ that start at the i^{th} bit position, where $i \geq 0$. For example, if $R(110101) = 01001011110101\dots$ then $H(110101, 0, 3) = 010$ and $H(110101, 1, 4) = 1001$ and so on.

The following is the heuristic RSA key generation algorithm. Observe that it outputs s_1 , s_2 , and cnt in addition to what is output by *GenPrivatePrimes1*. Note also that p and q have the same specifications as in *GenPrivatePrimes1*.

GenPrivatePrimes2():

input: none

output: $W/2$ -bit primes p and q such that

$$p \neq q \text{ and } |pq| = W$$

1. while (TRUE) do:
2. generate s_1, s_2, c_2 to be random strings
 using *RandomBitString1*()
3. compute $c_1 = H(s_1, 0, W/2)$
4. set $cnt = 0$
5. while (TRUE) do:
6. $p = H(s_1 || s_2, \frac{cnt * W}{2}, W/2)$
7. if $p \geq 2^{W/2-1} + 1$ and p is prime
 then goto step 9
8. $cnt = cnt + 1$
9. compute $n' = (c_1 || c_2)$

10. solve for the quotient q and the remainder r
 in $n' = pq + r$
11. if q is not a $W/2$ -bit integer or if
 $q < 2^{W/2-1} + 1$ then goto step 2
12. if q is not prime then goto step 2
13. if $|pq| < W$ or if $p = q$ then goto step 2
14. if $\gcd(e, \phi(pq)) = 1$ then goto step 15
15. set $S = (p, q, s_1, s_2, cnt)$
16. output S , zeroize all values in memory, and halt

The private key owner should always keep (p, q, s_2, cnt) secret. Observe that p can be reconstructed using s_1 and s_2 alone (cnt can be easily guessed).

Given N and s_1 , the derivation of the predetermined portion of n is publicly verifiable. This is accomplished by having the verifier (i.e., someone in possession of n and s_1) check that either $H(s_1, 0, W/2)$ or $H(s_1, 0, W/2) + 1$ equals the $W/2$ upper order bits of n . Here the $+1$ accounts for a potential borrow bit having been taken from c_1 in computing $n = n' - r = pq$.

Since the verifier performs this check on the upper order bits, the black-box device that outputs p and q must provide a proper pre-image s_1 under H for the upper order bits of n or else the verifier will assume that the device is faulty. The algorithm uses the subliminal channel in pq to “close” the channel in pq , and therefore constitutes a heuristic method for foiling high-bandwidth SETUP attacks. However, whereas n can be published, s_1 should not be published since it now contains a subliminal channel (see subsection 5.4).

The seed s_2 is for the benefit of the owner of p and q . A heuristic way to check that p was generated using H is to verify that the prime p is equal to $H(s_1 || s_2, \frac{cnt * W}{2}, W/2)$ and that a smaller value for cnt does not make $H(s_1 || s_2, \frac{cnt * W}{2}, W/2)$ a prime that is greater than or equal to $2^{W/2-1} + 1$. The goal is to force devices that implement this algorithm to

first commit to s_1 in the upper order bits of n and then commit to s_2 in the prime p .

This method is not perfect since it is feasible to leak a small number of bits in practice. This small channel can be implemented by matching bits in the generated modulus to some subliminal message, and rejecting moduli in which the matching fails. This brute-force method may be able to leak 8 bits (give or take) in practice. So, a very low bandwidth kleptographic attack is still possible.

The practical impact of this heuristic is that it significantly minimizes the capacity of the subliminal channel in composites by employing the channel to display a commitment based on a hash function. This algorithm can be deployed in existing software systems without having to redesign the key generation process. However, an alternative method exists that provides stronger security guarantees at the cost of significantly redesigning the key generation process. This is the subject of Section 6.

5.3 Output Distribution of Oracle-Based Key Generation

Under the assumed existence of a random oracle R , *GenPrivatePrimes2* produces primes p and q from the same set and probability distribution as *GenPrivatePrimes1*. The reason why this is so is the subject of this section. This demonstrates that *GenPrivatePrimes2* produces “normal” RSA primes, despite its awkward appearance.

Consider the division of n' by p that leads to the quotient q and the remainder r . The quantity $n' = pq + r = (2^{W/2-1} + 1)^2 + 0$ is the smallest possible value for n' such that $p, q \geq 2^{W/2-1} + 1$. But this equals $2^{W-2} + 2^{W/2} + 1$ and is thus a $(W - 1)$ -bit quantity. The greatest possible $W/2$ -bit integer is $p = q = 2^{W/2} - 1$. So, $n' = pq + r = pq + (p - 1) = (2^{W/2} - 1)^2 + (2^{W/2} - 1) - 1$ is the greatest possible

value for n' such that p and q are $W/2$ -bit integers. But this equals $2^W - 2^{W/2} - 1$ and is hence a W -bit quantity. This shows that all possible values for $n' = pq + r$ where $|p| = |q| = W/2$ and $p, q \geq 2^{W/2-1} + 1$ are contained in $\{0, 1\}^W$. It follows that for every pair (p, q) such that $|p| = |q| = W/2$ and $p, q \geq 2^{W/2-1} + 1$, there are p values in $\{0, 1\}^W$ that when divided by p yield q as the quotient.

Clearly c_2 is a random $W/2$ -bit string. Under the random oracle assumption c_1 is an independently random $W/2$ -bit string as well. Since all $n' \in \{0, 1\}^W$ are equally likely and p is an independently random prime under the random oracle assumption, step 11 chooses q uniformly at random from the $W/2$ -bit integers greater than or equal to $2^{W/2-1} + 1$ in C' . Those values for q that are composite will be rejected. Hence, q is chosen uniformly at random from all $W/2$ -bit primes.

This can be conceptualized as follows. The $W/2$ -bit prime p can be fixed and the space $\{0, 1\}^W$ can be “divided” into regions containing p numbers. A region consists of $qp + 0, qp + 1, qp + 2, \dots, qp + p - 1$. Observe that when each of these values is divided by p the quotient is q . Note also that there are p such values. The regions are the rows below.

...				
$(q-1)p+0$	$(q-1)p+1$	$(q-1)p+2$...	$(q-1)p+p-1$
$(q+0)p+0$	$(q+0)p+1$	$(q+0)p+2$...	$(q+0)p+p-1$
$(q+1)p+0$	$(q+1)p+1$	$(q+1)p+2$...	$(q+1)p+p-1$
...				

All regions of interest fall fully within the set of W -bit integers since no region is “cut-off” by the $2^W - 1$ upper limit. A *dart* can then be thrown at $\{0, 1\}^W$. Which region if any the dart lands in determines the value of q . Due to acceptance/rejection no region is

preferred and exactly one valid region must be selected.

This establishes that p and q are chosen, just before step 13 in *GenPrivatePrimes2* is executed, from the same set and probability distribution as p and q just before step 7 in *GenPrivatePrimes1* is executed. The remaining steps are essentially the same in both algorithms.

5.4 Security Analysis of Oracle-Based Key Generation

Observe that it is perilous for the key owner to publish s_1 along with $n = pq$ in regards to the output of *GenPrivatePrimes2*. The problem with publishing s_1 is that s_1 is a subliminal channel by itself. To see why this is a problem, suppose that the key owner publishes s_1 . Consider the following attack. The device can choose a seed for a pseudorandom number generator and use the resulting pseudorandom bit sequence to derive s_2 . The seed is asymmetrically encrypted using the insider's public key and s_1 assumes the value of the resulting ciphertext. Since s_1 is published, the insider obtains it, decrypts it, and computes s_2 using the resulting plaintext. The value s_2 allows $n = pq$ to be factored.

As a result of this subliminal channel, only the certification authority and the private key owner should know s_1 . A CA that is given (n, s_1) can check that the upper order bits of n is a proper commitment of s_1 .

It is claimed that the oracle-based RSA key generation heuristic, when implemented in a black-box, has the following trade-offs.

1. (*advantage*) It is difficult for the black-box device to generate a pair of primes (p, q) of a particular form (e.g., p having a long sequence of binary zeros). This results from the fact that the private

key holder can verify that the upper order bits of n correspond to a commitment of s_1 , and p is a verifiable commitment of s_2 .

2. (*advantage*) It is difficult for the black-box to display subliminal information in $n = pq$. Given (n, s_1) , a verifier can check that the upper order bits of n correspond to a commitment of s_1 . The method is not perfect, since a small number of bits can still be subliminally displayed in n (see subsection 5.2).
3. (*advantage*) The algorithm is self-contained. Users perform key generation without interacting with a verifier. It is compatible with existing RSA systems and optionally permits heuristic verification of public keys by a third party.
4. (*disadvantage*) The algorithm is slower than straightforward RSA key generation. *GenPrivatePrimes1* will test $O(\log n)$ candidates for p and $O(\log n)$ candidates for q . *GenPrivatePrimes2* performs two-level prime finding. It will test $O((\log n)^2)$ candidates for p before finding a satisfactory q . The reason for this is that if q is invalid then p is chosen all over again.

Advantage (2) foils the possibility of high-bandwidth secretly embedded trapdoor attacks.

It is important to emphasize that a black-box implementation of *GenPrivatePrimes2* does not eliminate the possibility of backdoor attacks. The attack mentioned in Section 1 that uses a nefarious secret seed for a PRNG is still possible. This type of backdoor can be implemented in *RandomBitString1()*, for instance. This attack is not as robust as a SETUP attack, but is nonetheless effective.

For non-interactive key generation, one can try to separate the source of randomness from the key generation algorithm by using two separate devices. In this approach, one device generates the random bits

and the other device uses the random bits deterministically. A user might try to verify the operation of the deterministic portion of the key generation system by performing, say, 2^{24} key generations. However, the deterministic key generation device may in fact have a secret on-board random number generator that it uses as the source of randomness in a given invocation with probability $1/2^{40}$. So, the resulting “deterministic” key generator may in fact be a randomized algorithm that leaks the private key rather infrequently.

It then becomes a game for the manufacturer to see how many invocations he or she can compromise without being detected. It is a game because detecting the attack amounts to generating enough key pairs to reveal the true Byzantine behavior of the “deterministic” device. No matter how many tests the user decides to conduct, the manufacturer can always deploy a new device that lowers the probability of attack just a little bit more.

Fortunately, an alternative to oracle based RSA key generation exists that addresses this problem. The solution involves the use of a trusted third party during the generation of an RSA key pair. This is the subject of the next section.

6 A Provably Secure Protocol for Abuse-Free RSA Key Generation

In this section the protocol of Guajardo and Juels for generating RSA keys with verifiable randomness is outlined [14]. The approach is similar to other solutions for subliminal channel elimination since it uses a third party that contributes randomness to the final output values, which in this case are the primes p and q . Unlike Desmedt’s construction that relies on proving NP statements [8], this protocol utilizes several zero-knowledge sub-protocols and as a result is quite efficient.

However, it should be noted that there have been other protocols for RSA key generation as well. Franklin and Boneh gave efficient techniques for a group of users to jointly generate an RSA key pair [5]. At the end of the protocol, a new RSA modulus $n = pq$ is known to all users. However, nobody knows the factorization of n . The public encryption exponent e is publicly known and each user holds a share of the corresponding private exponent d . This allows the users to perform threshold decryption of RSA ciphertexts. The results are in the *honest but curious* threat model. In this model, it is assumed that the adversary is *passive* and follows the prescribed protocol. The adversary may record all available values and later try to determine secret information. It is assumed that the adversary will not alter any values except as required by the cryptographic protocol.

The remainder of this section is dedicated to the Guajardo-Juels protocol since it was specifically designed to guard against dishonest or otherwise incorrect RSA key generation with the single user in mind [14]. The protocol is called KEGVER, which stands for key generation with verifiable randomness. In this protocol the user and the CA (a trusted third party) generate a key pair for the user and the CA does not learn the private key. At the end of the protocol the CA (not necessarily the general public) is convinced that the key pair has the correct form, that it is randomly chosen, and that its randomness was influenced by the CA’s coin tosses. To prevent subliminal leakage it might be necessary to avoid publishing all of the user’s outputs except n (the proof transcripts might contain a subliminal channel).

In a nutshell, KEGVER operates as follows. The user and the CA conduct a cryptographic protocol that leads to the generation of two random integers, x and y . The protocol has the property that the CA influences the selection of x and y . Furthermore, the CA verifies that the CA influenced these values as expected. The values x and y are known to the user, but not to the CA. The user then computes an RSA com-

posite n and proves to the CA that n is a Blum integer of the form $p^r q^s$ with $r = s = 1$. The user also proves that p and q lie in the intervals $[x, x + \ell]$ and $[y, y + \ell]$, respectively, for some public parameter ℓ . This effectively proves that p is close to x and q is close to y , thereby severely hampering the user's ability to unilaterally "choose" the primes p and q . The value ℓ is chosen to be small enough so that the user is constrained in the construction of n , yet large enough to ensure that the user can find p and q in these intervals.

Guajardo and Juels indicated that this protocol can be conducted using two independently designed implementations that communicate with each other to foil SETUP attacks. For example, one peripheral device can act as the "user" and another peripheral can act as the trusted third party in the protocol. It is necessary that at least one of these two implementations be honest in order to detect an insider attack. This is a promising direction for efficiently protecting against insider abuse.

7 Conclusion

Various insider attacks against RSA key generation were addressed that potentially impact the privacy of encryption as well as the non-repudiability of digital signatures. Many of the existing protocols for proving properties of RSA public keys were cited, along with an open problem regarding Blum integers.

A stand-alone algorithm for generating RSA keys was presented in the random oracle model that is similar in flavor to the DSA parameter generation algorithm. It is claimed that this heuristic minimizes the bandwidth of the subliminal channel in RSA composites, and hence heuristically guards against high-bandwidth SETUP attacks against key generation. However, the method does not prevent all insider attacks. This algorithm should be regarded as orthogonal to (as opposed to replacing in any way) the effi-

cient protocol of Guajardo and Juels that shows how a verifier can be convinced of the random nature of the two RSA primes belonging to the prover.

8 Acknowledgments

Special thanks goes to Burt Kaliski, Markus Jakobsson, and Ari Juels for making corrections and for greatly improving the presentation of this work.

References

- [1] M. Agarwal, N. Saxena, N. Kayal. PRIMES is in P. Preprint, August 6, 2002.
- [2] E. Bach, J. Sorenson. Sieve Algorithms for Perfect Power Testing. In *Algorithmica*, vol. 9, pages 313–328, 1993.
- [3] M. Bellare, P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Conference on Computer and Communications Security*, V. Ashby (Ed.), pages 62–73, ACM, 1993.
- [4] M. Bellare, P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology—Eurocrypt '94*, A. De Santis (Ed.), LNCS 950, pages 92–111, Springer-Verlag, 1995.
- [5] D. Boneh, M. Franklin. Efficient Generation of Shared RSA Keys. In *Journal of the ACM*, vol. 48, no. 4, pages 702–722, ACM, 2001.
- [6] J. Boyar, K. Friedl, C. Lund. Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies. In *Journal of Cryptology*, vol. 4, no. 3, pages 185–206, 1991.

- [7] J. Camenisch, M. Michels. Proving in Zero-Knowledge that a Number is the Product of Two Safe Primes. In *Advances in Cryptology—Eurocrypt '99*, J. Stern (Ed.), LNCS 1592, pages 106–121, Springer-Verlag, 1999.
- [8] Y. Desmedt. Abuses in Cryptography and How to Fight Them. In *Advances in Cryptology—Crypto '88*, S. Goldwasser (Ed.), LNCS 403, pages 375–389, Springer-Verlag, 1988.
- [9] Y. Desmedt, C. Goutier, S. Bengio. Special Uses and Abuses of the Fiat-Shamir Passport Protocol. In *Advances in Cryptology—Crypto '87*, C. Pomerance (Ed.), LNCS 293, pages 21–39, Springer-Verlag, 1988.
- [10] W. Diffie, M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, vol. 22, no. 6, pages 644–654, 1976.
- [11] R. Gennaro, D. Micciancio, T. Rabin. An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products. In *Conference on Computer and Communications Security*, pages 67–72, ACM, 1998.
- [12] J. van de Graaf, R. Peralta. A simple and secure way to show the validity of your public key. In *Advances in Cryptology—Crypto '87*, C. Pomerance (Ed.), LNCS 293, pages 128–134, Springer-Verlag, 1987.
- [13] O. Goldreich, S. Micali, A. Wigderson. How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design. In *Advances in Cryptology—Crypto '86*, A. M. Odlyzko (Ed.), LNCS 263, pages 171–185, Springer-Verlag, 1986.
- [14] J. Guajardo, A. Juels. RSA Key Generation with Verifiable Randomness. In *Public Key Cryptography—PKC '02*, D. Naccache, P. Paillier (Eds.), LNCS 2274, pages 357–374, Springer-Verlag, 2002.
- [15] R. Kumanduri, C. Romero. Number Theory with Computer Applications, Algorithm 9.2.9. page 229, Prentice Hall, 1998.
- [16] A. K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology—Asiacrypt '98*, K. Ohta, D. Pei (Eds.), LNCS 1514, pages 1–10, Springer-Verlag, 1998.
- [17] M. Liskov, R. Silverman. A Statistical Limited-Knowledge Proof for Secure RSA Keys. Submitted to IEEE P1363 working group.
- [18] National Institute of Standards and Technology (NIST). FIPS Publication 186-2: Digital Signature Standard (DSS), 2000.
- [19] J. M. Pollard. Theorems on Factorization and Primality Testing. In *Proceedings of the Cambridge Philosophical Society*, vol. 76, pages 521–528, 1974.
- [20] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, vol. 21, no. 2, pages 120–126, 1978.
- [21] R. D. Silverman. RSA Public Key Validation. In *Workshop on Cryptography and Computational Number Theory*, K. Y. Lam, I. Shparlinski, H. Wang, C. Xing (Eds.), Progress in Computer Science and Applied Logic, vol. 20, Birkhäuser, 2001.
- [22] G. J. Simmons. The Prisoners' Problem and the Subliminal Channel. In *Advances*

in Cryptology—Crypto '83, D. Chaum (Ed.), pages 51–67, Plenum Press, 1984.

- [23] G. J. Simmons. The Subliminal Channel and Digital Signature. In *Advances in Cryptology—Eurocrypt '84*, T. Beth, N. Cot, I. Ingemarsson (Eds.), LNCS 209, pages 364–378, Springer-Verlag, 1984.
- [24] M. E. Smid, D. K. Branstad. Response to Comments on the NIST Proposed Digital Signature Standard. In *Advances in Cryptology—Crypto '92*, E. F. Brickell (Ed.), LNCS 740, pages 76–87, Springer-Verlag, 1992.
- [25] A. Young. Kleptography: Using Cryptography Against Cryptography. PhD Thesis, Columbia University, 2002.
- [26] A. Young, M. Yung. The Dark Side of Black-Box Cryptography, or: Should We Trust Capstone? In *Advances in Cryptology—Crypto '96*, N. Koblitiz (Ed.), LNCS 1109, pages 89–103, Springer-Verlag, 1996.
- [27] A. Young, M. Yung, The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In *Advances in Cryptology—Crypto '97*, B. S. Kaliski Jr. (Ed.), LNCS 1294, pages 264–276, Springer-Verlag, 1997.

Secure Verification of Location Claims*

Naveen Sastry

Umesh Shankar
UC Berkeley

David Wagner

Abstract

With the growing prevalence of sensor and wireless networks comes a new demand for location-based access control mechanisms. We introduce the concept of secure location verification, and we show how it can be used for location-based access control. Then, we present the Echo protocol, a simple method for secure location verification. The Echo protocol is extremely lightweight: it does not require time synchronization, cryptography, or very precise clocks. Hence, we believe that it is well suited for use in small, cheap, mobile devices.

1 Introduction

Computer scientists are used to studying access control mechanisms where one's identity determines what one is authorized to do. However, in the physical world, identity is not the only thing that matters: often, the requester's physical location also plays an important role in determining access rights. This suggests studying location-based access control.

Location-based access control in the physical world is easy, natural, and familiar. For example, being able to turn on or off the lights in a particular room using traditional technologies *requires* having a physical

presence in the room. The very design of the light switch is what enforces the security policy. In contrast, achieving the same kind of guarantee with information systems, such as wireless networks, is less straightforward; it is not simply a matter of putting a switch in the right place. To enforce location-based access control policies on information resources, we need a way to perform *location verification*, where a principal's location is securely verified to meet certain criteria, e.g., being inside a particular room or a specific building.

Location verification enables location-based access control. After verifying a principal's location using a location verification protocol, the principal can be granted access to a particular resource according to the desired policy. This approach is naturally combined with physical security; guards or locks might be used to determine who is allowed to enter a building, then location verification employed to allow wireless access to all those inside. The location verification problem is the key technical challenge that must be surmounted to implement location-based access control.

Location-based access control has several benefits. Most importantly, it is natural for many applications. One simple policy might allow wireless control of only the lights for the room you are in, or might insist that a company server cease operating if it is taken outside the building. In addition, using location for access control obviates the need to establish shared secrets in advance: visitors to a building need not obtain wireless encryption keys prior to their visit.

*This work was supported in part by DARPA NEST contract F33615-01-C-1895, NSF CCR-0113941, and an equipment donation from Intel.

In this paper, we study the location verification problem. First, we introduce and define the location verification problem (Section 2). Then, we propose a new protocol for location verification, called *the Echo protocol* (Section 3), and we prove its security (Section 4). Additionally, we discuss the privacy implications of the Echo Protocol, observing that privacy is largely an orthogonal issue. This work provides a foundation for securely using location in wireless information systems.

2 Goals and Assumptions

2.1 Problem Statement

There are many natural variants of the secure location problem. We focus on solving the *in-region verification* problem: a set of *verifiers* V wish to verify whether a *prover* p is in a region R of interest. R may be a room, a building, a stadium, or other physical area. The region typically has some sort of physical control to restrict people's entry into it; the purpose, then, is to control access to resources that are not intrinsically constrained by physical security, such as wireless networks. The verifier infrastructure V may, in some cases, be a distributed system consisting of multiple nodes.

The protocol must run correctly in the face of adversaries. Thus, when p does not in fact have a physical presence inside R , the verifier must be careful not to accept p 's claim to be in R . Furthermore, if p does have a presence in R , the verifier should accept p 's claim; otherwise the protocol would not be useful in practice. We therefore require the following two properties to ensure that the protocol is useful and secure:

- **Completeness:** If p and V both behave according to the protocol, and p is in R , then V will accept that p is in R .

- **Security:** If V behaves according to the protocol and accepts p 's claim, then p , or a party colluding with p , has a physical presence in R .

It is important to distinguish between the problem we are addressing, the *in-region verification* problem, and the *secure location determination problem*. In the latter problem, V attempts to securely discover the physical location of p . In contrast, in the *in-region verification* problem, p claims to be in a particular region, and V accepts or rejects the claim. The prover's location claim serves as a hint for the verifier to confirm or disprove. Framing the problem in terms of secure *in-region verification*, not *secure location determination*, simplifies the problem and allows different location determination algorithms to be used.

In fact, it is possible to compose an *in-region verification* protocol with any location determination algorithm, even a potentially insecure one, without compromising the security of the ultimate guarantee that a prover is in the region. The *in-region verification* algorithm verifies whether the claimed location is in R or not; thus, p can use an insecure localization algorithm to generate a claimed location that will be securely tested for accuracy by V . At worst, p 's claim will be rejected; in no case will V believe something about p 's location that has not been securely checked. The prover p thus has the flexibility to choose any appropriate location determination algorithm, even if it has not been proven secure. After running the determination algorithm, p will know which claims it can plausibly make.

2.2 Assumptions

It is worth considering in more detail what our particular protocol is and is not attempting to do:

- **Regions, not points.** We are not attempting to verify the exact location of the prover. In

other words, the location claims we verify are not claims of particular *point* locations (plus or minus some error distance), but rather just presence in a particular region R of interest. This model accords well with our anticipated applications. We assume that before the verification protocol begins, both the prover and verifier know the definition of the region R .

- **Only “local” regions.** It is not a requirement to verify *all* location claims. More specifically, we only attempt to verify location claims for regions R that are “near” V . We will explore more precisely what this means in Sections 3 and 4. The restriction makes sense in light of the proposed application domain: controlling access to wireless resources once physical access to an area has been granted.
- **RF and sound capability.** The verifier and prover must each be able to communicate using both radio frequency (RF) and sound (typically ultrasound frequencies). We will use both transmission media in our protocol.
- **Bounded processing delay.** The prover must be able to bound its processing delay. We will describe the effects that a loose bound will have on the protocol in Section 4.

2.3 Threat Model

In order to verify the security property, we must consider the protocol with respect to a particular threat model. We assume the verifier nodes are all trusted, and they can communicate securely amongst themselves. In contrast, the prover p might behave maliciously, and we will consider an adversarial prover consisting of multiple colluding nodes, arbitrary computing power, and secure RF (speed of light) communication amongst its own nodes as well as sound generation and detection capability on each of its nodes.

Each adversarial node can generate directional signals. Furthermore, the verifiers will not be able to detect the presence of an adversary by monitoring the RF communications since an adversary can easily use encryption or send its data on different RF frequencies.

Lastly, by definition, the adversary must not actually have any presence in the region R . Otherwise, it would be able to make a legitimate claim and would not need to attack the protocol.

2.4 Design Principles

We designed our protocol according to the following design principles:

- **Make few resource demands on the prover and verifier.** We would like to minimize the computational power and hardware resources necessary to participate in the protocol. The real goal is to enable location proofs for a large class of devices.
- **No prearranged setup required.** It should not be necessary for the prover to have previously engaged in a setup or registration step with the verifier. This excludes many cryptographic solutions; even public-key cryptography requires prearranged trust relationships, and thus is not suitable for our purposes. By eliminating the setup step, we are enabling access to resources to be granted based on physical presence alone.

In settings where keys have been previously set up, we can use them to complement our protocol. In the full version of this paper[14], we discuss a variant of the Echo protocol where a challenge-response protocol can be used to verify that a particular principal is inside a given region.

- **Quantitative guarantees.** We would like to provide precise bounds on the uncertainty in the protocol.

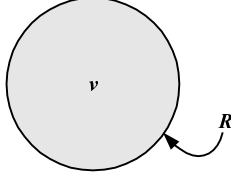


Figure 1: An illustration of our first simplification of the problem. The prover (not shown here) will try to convince the single verifier node v that it is inside the region R (depicted as a shadowed circle, which in this first scenario is assumed to be centered at v).

3 Our Design: The Echo Protocol

Next, we describe the design of our proposal for location verification, which we dub the Echo protocol. For expository purposes, we start by considering a simplified toy scenario and developing a simple protocol for this scenario (Section 3.1); then, we extend it repeatedly (Section 3.2) until we obtain the full protocol (Section 3.3).

Notation We define s to be the speed of sound, or 331 m/s. Likewise, we will take c to be the speed of light (which is the same as the speed of propagation of electromagnetic waves), or 3×10^8 m/s. Define $d(x, y)$ to be the distance between x and y . We define R to be the area in which we would like to verify the location of a prover p . The set of all verifier nodes is denoted by V . N denotes a nonce, i.e., an unpredictable random value.

3.1 Protocol Intuition

Consider first a simplified case, where we have only a single verifier node v , where the region R is a circle¹,

¹In practice, the region is a sphere, instead of a circle. This simplification makes the protocol easier to understand and does not affect the validity of our results.

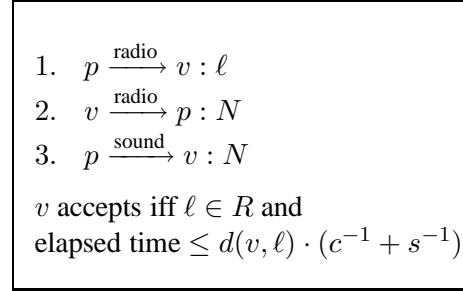


Figure 2: A protocol that solves our first simplification of the problem.

and where this circle is centered at v . This scenario is shown pictorially in Figure 1. Now, suppose that the prover claims to be at some location $\ell \in R$ inside the region.

We present a simple protocol for validating the location claim in this restricted case. First note that if the claimed location ℓ is not inside R , then the verifier can reject the claim immediately. Thus, we may safely assume that the prover claims to be inside R . Next, the verifier node v sends a packet containing a nonce to the prover using RF; the prover immediately echoes the packet back to the verifier using ultrasound. The verifier node v can then calculate how long it should take to hear the echo, namely, the sum of the time it takes to reach ℓ using RF, plus the time it takes for a return packet to go from ℓ to v using ultrasound. Thus, the total elapsed time for the prover to hear the echoed nonce should be about $d(v, \ell)/c + d(v, \ell)/s$ seconds. The only thing v has to do is time this process: If the elapsed time from the initial transmission to reception of the echo packet is more than this amount, the verifier node v rejects the prover's claim; otherwise, if the elapsed time is at most this expected amount, v accepts. This protocol is summarized in Figure 2.

Why does this work? If the prover is able to return the packet within some maximum amount of time, then the verifier is assured that the prover is within $d(v, \ell)$ meters of v . This means that ℓ is known to be inside a circle of radius $d(v, \ell)$ centered at v . Call this

circle C ; then we know $\ell \in C$. Since R is defined to be a circle of radius at least $d(v, \ell)$ centered at v , we have $C \subseteq R$, and hence $\ell \in R$. In short, we know that the prover must be inside R .

If the prover cannot return the nonce in sufficient time, it may be for one of two reasons. Either the prover is more than $d(v, \ell)$ meters away from v , or the prover has some processing delay between receiving the RF packet and returning the ultrasound packet. We will explore this latter issue in the following section.

What if the prover tries to cheat by delaying his response? This attack only increases the total elapsed time of the process, thereby making the verifier reject. Intuitively, the longer it takes to complete the protocol, the farther away the prover appears to be. It is not in the prover's interest to appear to be farther from v , because this will put the prover's apparent location outside of R , hence making v reject the prover's claim.

Can the prover cheat by starting the transmission of the response early? No, this attack is not possible. The nonce in the packet prevents the prover from sending a reply before it has received the outgoing RF packet. Hence, the speed of light and sound prevents the prover from pretending to be closer to v than he really is.

3.2 Processing Delay & Nonuniform Regions

In this section, we present a slightly more advanced protocol that addresses three additional issues: the fact that the prover has a nonzero processing delay, the fact that packets take nonzero time to transmit, and the fact that R might not be a circle. We base this protocol on the simple protocol presented in the previous section. For a more complete treatment of these considerations, please see the full version of this paper [14].

Processing delay So far we have assumed that a prover can immediately echo back the nonce it was sent. In reality, of course, there is some finite processing delay. Let us start with the configuration mentioned in Section 3.1: We have a single verifier located at the center of a circular region R . Suppose the prover can bound its processing delay to be at most Δ_p seconds and can make the verifier node aware of this maximum delay. Then, if the prover claims to be at ℓ , the verifier node can compute the time for a prover actually at ℓ to get the packet back: the time for the RF signal to travel from v to ℓ , a processing delay of at most Δ_p , and finally the time for the sound to travel from ℓ back to v . This creates a problem when the prover is near the edge of R ; the processing delay creates enough uncertainty that we cannot tell if the prover is inside or outside R . The solution is not to accept location claims such that the region of uncertainty lies outside R . Thus, we define the term *Region of Acceptance* (ROA) to be the area in which the verifier node v is sure that it can correctly verify claims for a prover. Note that this region depends on Δ_p . We write $\text{ROA}(v, \Delta_p)$ to indicate the region where location claims are permitted by v , if the claimed processing delay is Δ_p . See Figure 3 for an illustration.

An alternate way to view $\text{ROA}(v, \Delta_p)$ is that it is the region for which the protocol is complete.

What kind of processing delays do we expect to see? Our experiences with the Berkeley Mica 2 sensor network platform indicate that a millisecond delay is completely feasible and realistic [10]. Each device is a small, embedded device running an 8 megahertz, 8-bit Atmel 128 processor. Under most applications, the processor is idle most of the time to increase battery life, so system bus and processor contention is a minor contributor to delay. The operating system, TinyOS, is extremely small and simple, so the delay in the network stack are also minimized. We expect that most of the delay will arise in using a media access control protocol (MAC) to acquire the sending channel. By using an extremely simple MAC protocol that does

not include any waiting (sending immediately only if the channel is free), we can reduce both the variance and magnitude of the MAC delay. Each millisecond of delay contributes ≈ 33 centimeters of uncertainty. If a more powerful node is used, say comparable to a wireless base station, the delay could likely be reduced by an order of magnitude.

Packet transmission time Another subtle point in considering the security of the Echo Protocol is that transmitting a packet is not instantaneous since a sender sends the first bit of the packet and some time later finishes sending the last bit of the packet. An adversary can leverage this time differential if it is near the edge of the ROA. Under certain circumstances, an adversary can be outside the ROA, yet probabilistically convince the prover that it is inside, a violation of the security condition. An adversary that anticipates the first k bits of the nonce will gain an advantage since it can overlap the sending of the outbound packet while still receiving the incoming packet. By using a nonce generated by a cryptographically strong pseudo-random number generator, the verifier limits the effectiveness of this attack. An adversary has a 2^{-k} chance in correctly guessing k bits, so it is not feasible to anticipate more than a few bytes. As detailed in the full version of this paper [14], one solution is to incorporate the packet transmission time into Δ_p by increasing it by the maximum packet transmission time; we call the packet transmission time Δ_{\min} .

Non-circular regions. Up until now, we have been assuming that R is a circle centered at v . However, that is not always a realistic assumption: perhaps we are interested in verifying location claims in a square room, for instance. We will now relax that assumption and assume that the verifier node is contained somewhere within an arbitrarily shaped region R . This causes a larger area to be *incomplete*, or non-verifiable, as shown in Figure 4. We will address the question of incompleteness in the next section.

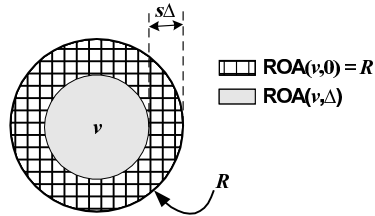


Figure 3: Diagram illustrating a single verifier at the center of a circular region R where there is an upper bound of Δ on the processing delay. The diagram illustrates the relationship between $\text{ROA}(v, \Delta)$ and $\text{ROA}(v, 0)$, the latter equal to R in this case.

Previously, $\text{ROA}(v, 0)$ had been equivalent to R . But this will not work when R is not a circle centered at v . Since we are assuming that our communications equipment is omni-directional and that signals travel at the same speed in all directions, the ROA must be a circle. Furthermore, the ROA must be wholly contained within R . By definition, the ROA is the region where the verifier will accept a correctly functioning prover; if the ROA were not fully contained within R , the prover could accept a location claim for a prover outside of R , which would be unacceptable. Furthermore, we would like to maximize the area of the ROA since a larger ROA leads to a larger coverage. Thus, $\text{ROA}(v, 0)$ should be the largest circle that fits within R ; in other words, it should be the largest circle that is tangent to R and still contained within it.

We now extend the protocol to handle non-circular regions R where the verifier can bound its processing delay to be at most Δ_p . Recall that both the prover node and verifier node know R *a priori*. Using this, the verifier node can compute ahead of time the region $\text{ROA}(v, 0)$.

The protocol then proceeds as follows: the prover first broadcasts its claimed location ℓ and processing delay Δ_p to the verifier. If $\ell \notin \text{ROA}(v, \Delta_p)$, the verifier should immediately reject the location claim since it will not be able to definitively validate the claim. Otherwise, the verifier node broadcasts a nonce to the

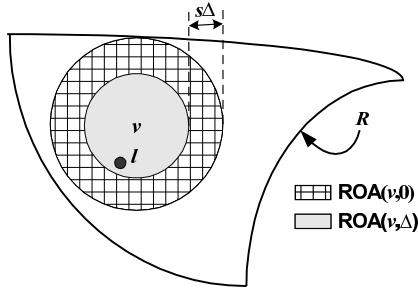


Figure 4: A single verifier v , inside a irregular region R . We are interested in proving that the prover is within R . The larger circle represents $\text{ROA}(v, 0)$, the area in which v is useful for location verification proofs. This is the largest circle centered at v and wholly contained within R . The inner circle represents $\text{ROA}(v, \Delta)$, the region in which v will accept location claims from a device that is able to bound its processing delay by Δ .

prover; the prover echoes the nonce back over ultrasound. The verifier can again time the communication: if it is no greater than the time for the signal to travel out and back and allowing for processing delay, the verifier accepts the location claim. Recall that Δ_p is an intrinsic property of the prover. So by sending Δ_p as the first step of the protocol, it can receive an early rejection if its delay is too large for its claimed location; thus, Δ_p is only useful for the prover. By lying about Δ_p , an adversary only affects early rejection and not the security of the protocol (see Section 4 for the complete proof of security). Intuitively, the total time that the verifier allows for a message to go out and come back is fixed and independent of Δ_p . So, when an adversary exaggerates Δ_p , it simply allows itself less time to respond. We expect that in practice if the prover were rejected early, the verifier would tell the prover $\text{ROA}(v, \Delta_p)$ so the prover could move into a verifiable area.

3.3 Full Protocol Description: The Echo Protocol

In the final iteration of the protocol, we introduce multiple verifier nodes in an attempt to increase the coverage of R . Recall that if R is not a circle, no single node can provide 100% coverage. Consequently, multiple verifiers are needed. Intuitively, we will run the protocol presented in Section 3.2 after selecting one verifier from among the set of verifiers V .

The protocol is quite simple. See Figure 6 for the complete definition. First, a verifier is chosen so that the claimed location ℓ lies within that verifier's ROA. If no such verifier exists, execution is aborted, since the claim can not be verified. After choosing a verifier v to participate, v sends a packet to p using RF, which is echoed back to it using ultrasound. v can calculate how long it should take to hear the echo, namely, the sum of the time it takes to reach ℓ using RF, plus Δ_p , plus the time it takes for a return packet to go from ℓ to v using ultrasound. If the measured elapsed time exceeds this anticipated time, v rejects the location claim. The nonce in the packet prevents the prover from sending a reply before it has received the outgoing RF packet.

The extra verifier nodes serve to expand the region of acceptance within R . Thus, while $\text{ROA}(v, \Delta_p)$ refers to the region that one particular verifier node can accept, we define $\text{ROA}(\Delta_p)$ to be the region where at least one verifier node can prove location claims. It is then clear that

$$\text{ROA}(\Delta_p) \equiv \bigcup_{v \in V} \text{ROA}(v, \Delta_p)$$

since the set of verifiers can accept a location proof if the claimed location is inside at least one verifier's region of acceptance.

In the Echo protocol, the infrastructure chooses a single verifier node to participate in the protocol. A verifier v may participate if $\ell \in \text{ROA}(v, \Delta_p)$, since

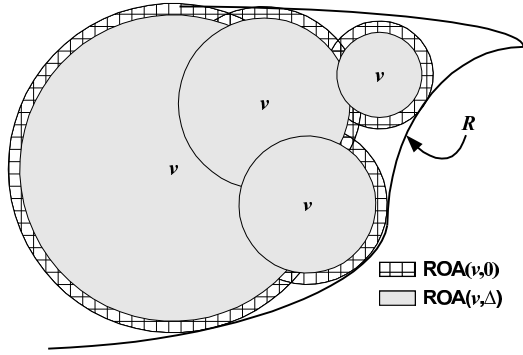


Figure 5: The relationship between $ROA(v)$ (for a single verifier v) and the aggregate ROA. Each gray circle represents $ROA(v, \Delta)$ for a particular verifier v . Taken collectively, the gray region represents $ROA(\Delta)$, the aggregate region in which the set of verifiers can successfully verify the location of a prover that features a processing delay less than Δ . Note that $ROA(\Delta)$ is wholly contained within R .

by definition that is the region for which it can perform secure location verification proofs. Note that the claimed location ℓ may be inside $ROA(v, \Delta_p)$ for many different verifier nodes v , hence more than one verifier node might be eligible for participation in the protocol. We only require one to be chosen, and we allow the verifiers to use any convenient leader election mechanism for choosing which particular verifier node will run the protocol. They may have a purely deterministic mechanism for electing verifiers, or they may use a dynamic algorithm in an attempt to conserve power, for example.

4 Security Analysis

As explained in Section 3, the protocol relies on timing: the amount of time it takes to get a response from the prover bounds the prover's distance from the verifier. We now show that it is impossible for an adversary outside R to convince the verifier that it is in R .

COMMUNICATION PHASE:

1. $p \xrightarrow{\text{radio}} \text{broadcast} : (\ell, \Delta_p)$.
The prover broadcasts its claimed location ℓ and processing delay Δ_p .
2. $v : t_s \leftarrow \mathbf{time}()$.
 $v \xrightarrow{\text{radio}} p : N$.
A single verifier v starts its timer and responds with a random nonce.
We require $\ell \in ROA(v, \Delta_p)$ and $\Delta_p \geq \Delta_{\min}$.
If no such verifier exists or Δ_p is invalid, **abort**.
3. $p \xrightarrow{\text{sound}} v : N$.
 $v : t_f \leftarrow \mathbf{time}()$.
The prover echoes the nonce over ultrasound.
The verifier records the finish time.

VERIFIER COMPUTATION PHASE:

4. **if** sent nonce differs from received nonce
return false
5. **if** $t_f - t_s > \frac{d(v, \ell)}{c} + \frac{d(v, \ell)}{s} + \Delta_p$
return false
6. Otherwise, **return true**

Figure 6: Formal description of the Echo protocol, which can perform location verification in an arbitrary region R with multiple verifier nodes. We represent the prover node as p and the verifier node that runs the protocol as v . In step 2, Δ_{\min} represents the lower bound on the delay, which is incurred by transmitting the packet.

Proof of security The heart of the argument is that an attacker would not be able to get the sound signal to the verifier in time. In order to confirm that the prover is at ℓ , all a particular verifier node v must do is verify that the incoming sound signal, which includes the outgoing nonce, is received within

$$t_{\max} \leq \frac{d(v, \ell)}{c} + \frac{d(v, \ell)}{s} + \Delta_p \text{ seconds,}$$

where $d(v, \ell)$ is the distance from the verifier to the claimed location, c is the speed of radio propagation (the speed of light, which may vary depending on

the medium through which it passes), s is the speed of sound, and Δ_p is the prover's processing delay. As described in Section 3.2, Δ_p includes the packet transmission time. This is checked by the verifier in step two of the communication phase of the protocol. Recall that v agrees to run the protocol only if $\ell \in \text{ROA}(v, \Delta_p)$, i.e., if the circle of radius $d(v, \ell) + \Delta_p \cdot s$ lies wholly within R . As long as Δ_p is positive (guaranteed by step two), we know that $\text{ROA}(v, \Delta_p) \subset R$. As we saw in Section 3.2, Δ_p is used as an optimization and potentially a hint to the prover, so even if an adversary lies about Δ_p , security is assured.

By definition, the attacker A is outside R ; thus we have

$$d(v, A) > d(v, \ell) + \Delta_p \cdot s.$$

Let Δt^A denote the elapsed time ($t_f - t_i$) when the attacker finishes sending its response (message 3 of the Echo protocol). The attacker has only two choices: either guess at least some of the bits of N , or learn the entire nonce N from v . In the former case, the attacker's success probability can be made negligibly small by choosing N from a set of sufficient size. In the latter case, it will take at least $d(v, A)/c$ seconds after v first reveals N before A can receive N , because no signal can travel faster than the speed of light. Because v reveals N for the first time in message 2 of the protocol, $\Delta t^A \geq d(v, A)/c$ in this case. Now, since the attacker cannot finish transmitting its response before it has received the entire nonce, and because the attacker's response cannot travel faster than the speed of sound, the minimum time required for the attacker

to hear N and get a response to v is

$$\begin{aligned} \Delta t_{\min} &= \Delta t^A + \frac{d(v, A)}{s} \\ &\geq \frac{d(v, A)}{c} + \frac{d(v, A)}{s} \\ &> \frac{d(v, \ell) + \Delta_p \cdot s}{c} + \frac{d(v, \ell) + \Delta_p \cdot s}{s} \\ &\geq \frac{d(v, \ell)}{c} + \frac{d(v, \ell)}{s} + \frac{\Delta_p \cdot s}{c} + \frac{\Delta_p \cdot s}{s} \\ &\geq \frac{d(v, \ell)}{c} + \frac{d(v, \ell)}{s} + \Delta_p. \end{aligned}$$

Consequently, the attacker's signal cannot reach the verifier before the deadline. Note that nowhere in our analysis did we rely on *which* verifier node was used. The only difference would be in the magnitude of the error terms and, therefore, in the chance that the location claim would even be accepted for verification. The attacker does not gain any advantage by selecting a different verifier from the one selected to participate.

Attacks One possible attack could exploit the difference in propagation speed of sound in different media. For example, the speed of sound in steel is 5032 m/s, nearly 15 times faster than in air; other materials exhibit similarly higher sound transmission speeds than air. If the verifier's estimation of s is slower than the actual one, then the proof above does not apply. If this is a valid threat model—say there is a lot of metal near the verification region that is capable of transmitting sound from the outside—then the verifier's estimation of s should be adjusted. This can be done once on a site-specific basis. An alternate defense would be to have other verifier nodes confirm the estimate of s based on when the sound signals are received.

More generally, we require that there be no way for an attacker to generate sound waves from afar without being subject to speed-of-sound delays. For instance, a remote attacker could call up some person in R over the telephone and convince the victim to put the call

on speakerphone, then run the protocol. If the ultrasound reply can go over the telephone with sufficiently high fidelity, then the attacker might be able to spoof his location. The key is that the attacker has evaded the speed-of-sound limit on signal propagation by exploiting the ability to remotely actuate a loudspeaker located inside R . We expect such “remote actuation” attacks will be very difficult to mount in practice; in our example, band-limited phones would block ultrasound.

Variants We Rejected One might also consider the implications of other variants of the protocol, where the use of sound and radio for the outgoing and incoming signals is changed from (radio, sound) to (radio, radio), (sound, radio), or (sound, sound). If radio communication is used in both directions, then the error term $\Delta \cdot c$ would be very large (10^5 to 10^6 times as large as the sound case), and it is quite likely that the verifier would not accept location claims at all, since the error might exceed the size of R itself! Thus, at least one of the two directions should use sound.

Why did we reject (sound, radio)? There is a subtle attack. If sound is used in the outgoing direction, an attacker might be able to break security by using laser-based remote “bugging.” The trick is to bounce a laser off a window within R and analyze the return signal to detect the vibration of the window, which would allow a sophisticated attacker outside R to “bug” a room within R from miles away without being subject to speed-of-sound delays on the propagation of the sonic signal. Thus, “remote bugging” attacks effectively speed up the transmission speed of the sound wave and thereby invalidate our security proof above. We thus reject (sound, radio) and (sound, sound) since both rely on transmitting sound in the outgoing direction.

The (radio, sound) protocol is more secure against such attacks, because “remote actuation” seems significantly more difficult than “remote bugging,” and

the security of the (radio, sound) protocol rests only on the difficulty of “remote actuation” and not on the hardness of “remote bugging.” For this reason, the Echo protocol uses radio in the outgoing direction and reserves ultrasound for the return signal from the prover.

Privacy Implications We now look at privacy concerns related to the Echo Protocol. Clearly, the prover reveals information to the verifier – it’s trying to convince the verifier that it is in the ROA, after all. The prover does have some control over what the verifier learns, however. A prover situated very close to the verifier can wait some time before replying with its nonce. This increases the verifier’s uncertainty about the prover’s location. A prover can employ this technique to ensure that the verifier only learns that the verifier is inside the ROA centered around it.

But what about an outside observer? A shareholder snooping at a company’s headquarters might expect a press release if ten vice presidents each authenticate their location to the corporate boardroom. An adversary watching the interactions between the prover (vice president) and the room’s infrastructure could infer the prover’s location. In fact, since in most cases the verifier is non-malicious, an adversary could infer a prover’s location just by knowing which verifier the prover interacts with. However, we note that it is easy enough to obtain the prover’s location via other means: for example, an adversary can passively triangulate a node’s ordinary wireless communications and determine its location independent of any location protocol [17].

If an adversary can find a prover passively using triangulation, why can’t the verifier use triangulation? An adversary can inexpensively break the security of a system that uses triangulation. By appropriately sending different signal strengths to each verifier using directional antennas, an adversary can foil a triangulation system to create a ghost image at any location.

This highlights a difference in the goals of the adversary and verifier: an adversary still “wins” if it can successfully triangulate only 1% of the time. However, the verifier must only accept claims in accordance with the security condition, so it cannot use an unreliable approach.

5 Related Work

Other Approaches A number of authors have proposed using time-of-flight measurements and the speed of light to securely gain location information about untrusted parties. Brands and Chaum proposed a time-bounded challenge-response protocol [4] as a defense against man-in-the-middle attacks on cryptographic identification schemes. Hu, et al., proposed using temporal packet leases for wireless networks to defend against similar attacks [11]. However, a major limitation of these schemes is that both the prover and verifier send RF signals, requiring a much more accurate timing system at the verifier as well as tight real-time processing guarantees on both the prover and verifier for accurate readings. For these reasons, we believe our algorithm is better suited to mobile devices than those previous proposals.

In independent and concurrent work, Waters and Felten present a scheme that uses round-trip time-of-flight of RF signals to achieve goals similar to ours [21]. Their architecture is similar to ours, in that they, too, suggest focusing on secure location verification rather than on secure location determination. However, their reliance on RF seems likely to limit deployment, like the previous proposals mentioned above. Additionally, by using tamper-resistant trusted devices, they are able to defend against stronger adversaries. If their verifier accepts, they can successfully show that the trusted device is at the specified location. In comparison, we can show that the device or a collaborator has a presence at the specified location.

Vora and Nesterenko use a novel technique for the secure location verification problem that doesn’t rely on time of flight[19]. The intuition behind their idea is simple: nodes nearby the prover’s claimed location should hear a prover’s broadcast, while those further away should not hear it. They make use of “rejector nodes” to monitor radio signals from a malicious verifier node that is outside the ROA. When a verifier claiming to be inside the ROA broadcasts and a rejector node hears the signal, the system does not accept the verifier’s claim. Their scheme is, however, vulnerable to adversaries with directional antennas and requires very careful node placement to handle non-trivial radio falloff models.

Location & Localization The idea of using time-of-flight to estimate distance is not a new one: it dates back to the birth of radar systems, which often use time-difference-of-arrival (TDOA) to determine the range to detected objects. Ultrasonic time-of-flight ranging can even be found in nature, where it is used by bats.

Coarse-grained location authentication has been used in the television industry to prevent cloning of set-top boxes [8]. Gabber and Wool propose four coarse-grained techniques, relying on extensive telecommunications infrastructure such as satellites, paging and cellular networks. Their techniques rely on tamper-resistant hardware.

Location-limited channels provide a communication mechanism that is restricted to a short range and provides both endpoints a mechanism to guarantee the authenticity of each participant [16]. Balfanz, et al., have proposed using location-limited channels for location-based access control [3], and many others have also proposed use of limited-range radio broadcasts as a way to verify proximity [5, 6, 12]. However, there are no strong security guarantees that the communication range will always be limited as desired: an adversary with more powerful equipment may be able

to participate in the protocols even if they are substantially further away than non-malicious parties.

Finally, there are many techniques to help localize devices [1, 2, 15, 13, 9, 20], GPS being one of the most widely deployed. However, none of those works addresses security, and in fact GPS signals can be spoofed [18, §3.2.2]. Nonetheless, we have noted that combining a (possibly insecure) localization mechanism with our secure location verification technique yields a secure localization algorithm. Thus, insecure localization protocols should be seen as complementary to our work on secure location verification.

Many authors have commented on the value of location-based access control [5, 6, 7, 12, 3].

6 Conclusion

We introduced the in-region verification problem. Then, we designed a provably secure, lightweight protocol to address it, named the Echo protocol. The Echo protocol does not require cryptography, time synchronization, or any prior agreement between the prover and verifier, making it suitable for low-cost devices such as those in sensor networks. It is robust against a malicious adversary with unbounded computing power; the security rests on physical properties of sound and RF signal propagation. We expect the Echo protocol to be a useful contribution in contexts where physical presence is used for access control.

References

- [1] GPS Documentation. https://www.peterson.af.mil/GPS_Support/gps_documentation.htm.
- [2] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *INFOCOM (2)*, pages 775–784, 2000.
- [3] Dirk Balfanz, D.K. Smetters, Paul Stewart, and H. Chi Wong. Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In *Network and Distributed System Security Symposium Conference Proceedings*, 2002.
- [4] Stefan Brands and David Chaum. Distance-Bounding Protocols. In *EUROCRYPT '93*, volume 765 of *LNCS*.
- [5] Deborah Caswell and Philippe Debaty. Creating Web Representations for Places. In *2nd International Symposium on Handheld and Ubiquitous Computing*, pages 114–126, 2000.
- [6] Mark D. Corner and Brian D. Noble. Zero-Interaction Authentication. In *MOBICOM '02*. ACM Press, 2002.
- [7] Dorothy E. Denning and Peter F. MacDoran. Location-Based Authentication: Grounding Cyberspace for Better Security. In *Computer Fraud & Security*. Elsevier Science Ltd., February 1996.
- [8] Eran Gabber and Avishai Wool. How to Prove Where You Are: Tracking the Location of Customer Equipment. In *Proceedings of the 5th ACM conference on Computer and Communications Security*, pages 142–149, 1998.
- [9] Lewis Girod, Vladimir Bychkovskiy, Jeremy Eison, and Deborah Estrin. Locating Tiny Sensors in Time and Space: A Case Study. In *ICCD*, 2002.
- [10] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *ASPLOS*, 2002.

- [11] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *INFOCOM*, 2003.
- [12] Tim Kindberg, Kan Zhang, and Narendar Shankar. Context Authentication Using Constrained Channels. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, 2002.
- [13] A.M. Ladd, K.E. Bekris, G. Marceau, A. Rudys, D.S. Wallach, and L.E. Kavraki. Robotics-Based Location Sensing for Wireless Ethernet. In *Eighth Annual International Conference on Mobile Computing and Networks (MobiCOM 2002)*, 2002.
- [14] Naveen Sastry and Umesh Shankar and David Wagner. Secure verification of location claims. In *ACM Workshop on Wireless Security (WISE-2003)*, 2003.
- [15] Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth J. Teller. The cricket compass for context-aware mobile applications. In *Mobile Computing and Networking*, pages 1–14, 2001.
- [16] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks. In *7th Security Protocols Workshop*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–92, 1999.
- [17] Ping Tao, Algis Rudys, Andrew Ladd, and Dan Wallach. Wireless LAN location sensing for security application. In *ACM Workshop on Wireless Security (WISE-2003)*, 2003.
- [18] John A. Volpe. Vulnerability Assessment of the Transportation Infrastructure Relying on the Global Positioning System, August 2001.
- [19] A. Vora and M. Nesterenko. Secure location verification using radio broadcast. <http://www.cs.kent.edu/~mikhail/Research/location.ps>, submitted to 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks.
- [20] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4(5):42–47, October 1997.
- [21] Brent Waters and Ed Felten. Proving the Location of Tamper Resistent Devices. http://www.cs.princeton.edu/~bwaters/research/location_proving.ps.

Manual authentication for wireless devices

Christian Gehrman
Ericsson Mobile Platforms
Lund, Sweden.
christian.gehrmann@ericsson.com

Chris J. Mitchell
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK.
c.mitchell@rhul.ac.uk

Kaisa Nyberg
Nokia Research Center
Helsinki, Finland.
kaisa.nyberg@nokia.com

January 23, 2004

Abstract

Manual authentication techniques have been designed to enable wireless devices to authenticate one another via an insecure wireless channel with the aid of a manual transfer of data between the devices. Manual transfer refers to the human operator of the devices performing one of the following procedures: copying data output from one device into the other device, comparing the output of the two devices, or entering the same data into both devices. Techniques currently being standardised are described which achieve this, and which require only small amounts of data to be transferred between the two devices. This makes the mechanisms particularly attractive for non-expert use, as required for ubiquitous mobile wireless devices.

1 Introduction

Entity authentication and authenticated key establishment are of fundamental importance in establishing secure communications between a pair of communicating parties. Entity authentication is normally pro-

vided when a communications link is established and, if an authenticated key is established simultaneously, this can be used to protect subsequently exchanged data. The purpose of this paper is to examine how these services might best be achieved for personal wireless-enabled devices.

Using the terminology of Stajano [12], the problem is that of securely ‘imprinting’ a personal device. That is, suppose a user has two wireless-enabled devices, e.g. a mobile phone and a Personal Digital Assistant (PDA); suppose further that he/she wishes the two devices to establish a secure association for their wireless communications. This will, for example, enable the two devices to securely share personal data. The problem is thus for the two devices to mutually authenticate one another and, where necessary, to establish a shared secret key, all using a wireless communications link. A shared secret key can be used as the basis for future secure communications between the two devices, including further mutual authentications.

The main threat to the process is via a so-called ‘man-in-the-middle attack’ on the wireless link. Because the link uses radio, a third party with a receiver and a powerful transmitter could manipulate the com-

munications between the devices, in a way that will not be evident to the user. Thus, the attacker could masquerade as the first device to the second device, and also as the second device to the first device, and set up separate keys with each. To prevent this, it will be necessary for the device operator to input and output data via the devices' user interfaces (i.e. performing a manual data transfer) to enable the devices to verify each other's identities.

This is the context of use for the manual authentication protocols described here. We make the following assumptions about the two devices.

- The two devices have access to a wireless communications channel, which can be used to exchange as much data as required; however, no assumptions are made about the security of this channel — for example, it may be prone to manipulation by an active attacker.
- The two devices are both under the control of either a single human user, or a pair of users who trust one another and who share a communications channel whose integrity is protected by some means (e.g. using handwritten notes or a voice channel). Both devices have a means to input or output a sequence of digits, i.e. they have at least a numeric keypad or a multi-digit display.
- If a device does not have a keypad, then it must at least have an input, e.g. a button, allowing the successful conclusion of a procedure to be indicated to the device. Similarly, if a device lacks a multi-character display, then it must at least have an output capable of indicating the success or failure of a procedure (e.g. red and green lights or a sound output).

We do not assume that the devices have any prior keying relationship or are equipped with any keys by their manufacturers. Of course, the problem would become dramatically easier if every device had a unique

signature key pair and a certificate for their public key signed by a widely trusted Certification Authority (CA). However the overhead of personalising every device in this way is likely to be prohibitive, particularly for low-cost devices.

Similarly, we do not assume that the two devices share a trusted communications link, e.g. as might be provided by a hard-wired connection. Such a link, even if it only guaranteed data integrity and data origin authentication (and not confidentiality), would again make the problem simple, since it could be used to authenticate a Diffie-Hellman exchange (as described in section 2). However, it would be unreasonable to always expect such a link to exist, since many simple wireless devices are likely to possess no wired communications interfaces.

An emerging international standard, ISO/IEC 9798-5 [6], currently at Committee Draft ballot stage, contains a set of 'manual authentication' solutions to the wireless device imprinting problem. Some of the schemes in this standard are described in sections 3 and 4 below. The same schemes may also be included in a future version of the Bluetooth standards. The existing Bluetooth specifications already contain a solution to device imprinting, but this solution has well-known security shortcomings if the initial exchange between devices can be wiretapped [3, 8]. A more detailed discussion of manual authentication can be found in [3].

2 Using Diffie-Hellman

Perhaps the most straightforward solution to the imprinting problem is to use the Diffie-Hellman key establishment protocol [1, 11]; this approach was first proposed by Maher [10]. As discussed in [3], this is also the solution proposed by Stajano and Anderson, [12, 13]. We thus first describe such a solution.

2.1 Procedure

The two devices first agree on (or are pre-programmed with) a secure set of Diffie-Hellman parameters, namely a large prime p , a large prime q dividing $p - 1$, and a value g of multiplicative order $q \bmod p$. In fact the Diffie-Hellman parameters could be standardised, or made the subject of an industry agreement. This would make the task of equipping all devices with the parameters very simple, and certainly much simpler and cheaper than giving each device an individual key pair and certificate.

The two devices, A and B say, then both generate a random value between 1 and $q - 1$ — call these values a and b . A computes $g^a \bmod p$ and sends it to B , and B computes $g^b \bmod p$ and sends it to A (in both cases using the wireless link). Finally A computes the shared key K as $K = (g^b)^a \bmod p$ and B computes the same key as $(g^a)^b \bmod p$.

Of course, as is widely understood (see, for example, [11]) this procedure does not provide mutual authentication, since the transmitted Diffie-Hellman values could have been manipulated by an interceptor acting as a man-in-the-middle (as above). Mutual authentication can be achieved by the manual exchange of checksums, i.e. using a ‘manual authentication’ technique, as follows. Note that, apart from the authentication issue, the security of the Diffie-Hellman protocol has been widely studied. If the parameters are chosen appropriately, then it is believed to be secure.

To provide the desired authentication, Maher [10] proposed the following additional steps. After completing the Diffie-Hellman exchange, both devices input the key K to a one-way hash-function h , e.g. SHA-1 [5], to obtain $h(K)$, which can be truncated to the desired length by taking the leftmost bits of the output (the choice for the length is discussed below). Suppose now that one device has a display and the other a keypad. The device with a display outputs the hash-

code, e.g. as a sequence of hexadecimal digits. The user now enters this sequence into the second device using its keypad. The second device compares the input hash-code with its computed value and, if they agree, provides a positive indication to the user. If this positive indication is received, the user inputs a success indication into the first device. This completes the mutual authentication process, and both devices also now have an authenticated key.

A similar procedure can be followed if both devices have a display. In this case both devices output their computed hash code, and the user is then simply required to compare the values output by the two devices. If they agree then the user gives a positive indication to both devices.

If one of the devices possesses neither a display nor a keypad, then it is difficult to apply the above method. However, in the case of a device with an audio output, e.g. a wireless headset, it may be possible for the headset to ‘speak’ the digits to the user, thus providing the functions of a display.

2.2 Issues

The main problem with the above procedure is the number of digits that need to be typed or examined by the user. Typing in a large number of digits to a small numeric keypad without making an error is a non-trivial procedure, and one that many users are likely to find too demanding to carry out. This is bad news for manufacturers of consumer devices. If the device cannot operate without completing the procedure then repeated failures to perform it correctly will cause the user to be very frustrated with the supplier. Alternatively, if the device can be used without a secure imprinting process, then this is a situation which could give rise to serious security vulnerabilities, which could also seriously damage the supplier’s reputation.

One possible solution is to drastically truncate the hash-code, e.g. to the first 16 or 32 bits — this would mean that the user would only have to type in (or compare) 4 or 8 hexadecimal digits, respectively. Whilst this is attractive, it has serious security weaknesses, as follows.

2.3 Attacking short hash-codes

Typically one device (say A) will send its Diffie-Hellman value first, and then wait for the response from B . Suppose that A first sends $g^a \bmod p$ to B . Suppose also that an active interceptor of the wireless link, C say, prevents this from reaching B and replaces it with $g^{a'} \bmod p$, for some value a' chosen by C .

B responds with $g^b \bmod p$, and B simultaneously computes the shared key as $K_B = g^{a'b} \bmod p$. Now suppose that C also intercepts $g^b \bmod p$ and prevents it from reaching A . It is important to observe that, because C chose a' , C is now able to compute the key held by B , i.e. C knows K_B .

C next generates a series of random values b' , and for each such value computes $K' = (g^a)^{b'} \bmod p$ and $h(K')$. C then compares the first 16 bits of $h(K')$ with the first 16 bits of $h(K_B)$. If they agree then C simply sends $g^{b'} \bmod p$ to A , who generates the key $K_A = (g^{b'})^a \bmod p$. Because of the way in which b' was chosen by C , the truncated hashes computed by A and B will match, although they do not share a secret key. Moreover, worst of all, C will know the values of the keys held by A and B .

This attack requires the attacker to perform a significant amount of work in a short time, i.e. before A and B ‘time out’. Specifically, if the hash-function is truncated to t bits, then on average C will need to generate 2^{t-1} values b' before one is found which yields the desired hash-value. Thus for $t = 16$, $t = 32$ and $t = 48$ the attack requires 30,000, 2 billion and 150 trillion

trials respectively. If an attacker with significant computing resources wished to attack the imprinting process, then it might be feasible to perform one billion trials in a second, and we might reasonably assume the ‘time-out’ value to be at most 10 seconds, allowing time for the attacker to perform 10 billion trials.

Thus using a hash-code of at least 48 bits appears to be necessary to rule out the possibility of success in attacking the imprinting process, given a well-equipped ‘man-in-the-middle’. 48 bits amounts to 12 hexadecimal digits, which is already quite a significant number for a user to enter in an error-free way, particularly when using a very small keypad with no display to enable the user to check the correctness of each key depression.

Ideally we would like a solution in which such a man-in-the-middle attack can be prevented without requiring the users to type in or compare long strings of digits. Such schemes form the focus of the remainder of this paper.

3 Manual authentication using a short check-value

We first describe an example of a scheme which uses keyed check-functions having short check-values (e.g. of around 16–20 bits) and using short keys (again of 16–20 bits). These check-functions are essentially MAC (Message Authentication Code) functions producing short outputs. To maximise the provable performance of the scheme, Gehrman and Nyberg [3] have proposed using a coding theory construction to compute the check-values; this scheme is included in the draft standard [6]. However, in practice, use of a conventional MAC function (e.g. a CBC-MAC based on use of a block cipher — see, for example, [11]) will almost certainly be sufficiently secure (in such a case the short key could be padded with a fixed string to construct a block cipher key).

3.1 The MANA I scheme

The scheme we describe, called MANA I (for MANual Authentication) in [3] and mechanism 1 in [6], is designed for use in the situation where one device (A) has a display and the other (B) has a keypad, although a simple variant (MANA II) exists for the case where both devices have a display. MANA I and MANA II were originally published in [2].

We also assume that the two devices wish to agree on the value of a public data string D . This data string could be the concatenation of A 's and B 's public keys, for some asymmetric cryptosystem. This could support the registration process for a small-scale PKI, or could simply be used as the basis for subsequent secure communications. In particular the public keys could be used, e.g. as Diffie-Hellman public keys, to provide the basis for an authenticated secret key establishment protocol, requiring no further intervention by the user.

We write $m_K(X)$ for the check-value computed using key K and data string X . The scheme operates as follows (see also Figure 1).

1. A data string D is agreed by some means between A and B using the wireless channel. This would typically occur via an exchange of (unprotected) messages.
2. Device A generates a random key K of length appropriate for use with the check-function (i.e. of 16–20 bits); A also generates the check-value $m_K(D)$. The key and check-value are then output to the display by device A .
3. The user enters the check-value and the key K , read from the display of device A , into device B (using the keypad).
4. Device B uses the key K provided by the user to recompute $m_K(D)$, and compares this with the

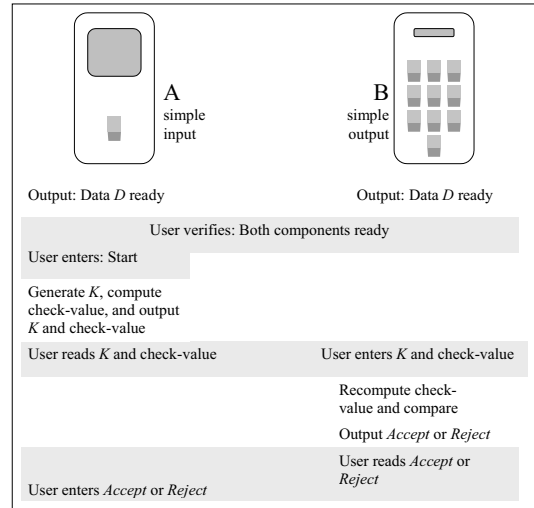


Figure 1: Manual authentication using a short check-value

value entered by the user. The device outputs an indication of success or failure, depending on whether or not the check-values agree.

5. The user copies this success/failure indication back into device A .

3.2 Analysis of the scheme

First note that the key and check-value are not available to any would-be attacker, who only sees the data D . The only possible strategy for the attacker is to try to persuade A and B to agree on different data strings D_A and D_B respectively, with the property that

$$m_K(D_A) = m_K(D_B)$$

for the largest possible number of secret keys K . In the coding theory construction for the check-function m (as in [3]) this largest number of keys is known. From it an upper bound on the success probability of

the attacker can be determined. However to carry out such a substitution attack, and to reach the largest success probability, would require a lot of computation. In practice an attacker cannot do significantly better than choose arbitrary strings D_A and D_B and hope that the check-values computed with the key that is unknown to the attacker will be the same. Even if a check-value construction based on a conventional CBC-MAC is used, it is very unlikely that the attacker will be able to do much better than with the coding theory construction, given that A and B choose parts of D_A and D_B , respectively. If a guess fails, no off-line computation can help the attacker any further. Once the parties have agreed on a key, it is no longer possible to attack the manual authentication scheme or to persuade either party to accept an incorrect value D . The only possible remaining vulnerability would be in any subsequent key exchange process based on use of the authenticated value D , e.g. a Diffie-Hellman key agreement (which, as we have discussed previously, is believed to be secure).

In summary, and assuming that the coding theory construction is used for the check-function, the probability of a successful attack (where A and B agree on different data values) is less than 2^{-13} , i.e. 1 in 8,000, for 16-bit keys and check-values and less than 2^{-17} , i.e. 1 in 130,000, for 20-bit keys and hash-codes. More details are given in [3, 6].

3.3 Variants of the scheme

Since the data string D is generated by A , it does not need to be sent to B until after step 2. In fact, the data transfer can be delayed indefinitely, and the key and check-value transferred manually to B can act as a ‘certificate’ for the subsequently exchanged data D . This might be useful, for example, where a newly purchased device is ‘manually authenticated’ as soon as it is switched on, but where public keys are only generated and exchanged at some later time.

Finally note that a variant of the above mechanism, known as MANA II in [3] and mechanism 2 in [6], can be devised to cover the situation where both devices A and B have a display, but neither of them has a keypad (although they must both possess a means of indicating successful completion of the protocol).

Briefly, in this case, the first two steps are as in MANA I. However, in addition to displaying the key and check-value, device A also sends the key to device B via the wireless channel (and hence in this case the key is available to an attacker). Device B uses the received key to recompute the check-value on its version of the data string, and finally displays the key received from A together with the check value it has computed. The user completes the process by comparing the values displayed by the two devices. Only if the key and check-value agree completely does the user give a ‘success’ indication to both devices.

4 Manual authentication using a MAC function

A different class of manual authentication protocols can be constructed using a conventional MAC function, such as HMAC [4] or a block cipher based CBC-MAC.

4.1 The MANA III scheme

The scheme we describe is MANA III from [3] (it is also specified as mechanism 3a in ISO/IEC CD 9798-6 [6]). It is designed for use in the situation where both devices have a keypad, although a simple variant exists for the case where one device has a display (see below). As previously, we assume that the two devices wish to agree on the value of a public data string D , where D could be used for agreeing public keys, as described in section 3.1.

We write $m_K(X)$ for the MAC computed using key K and data string X . The scheme operates as follows (see also Figure 2).

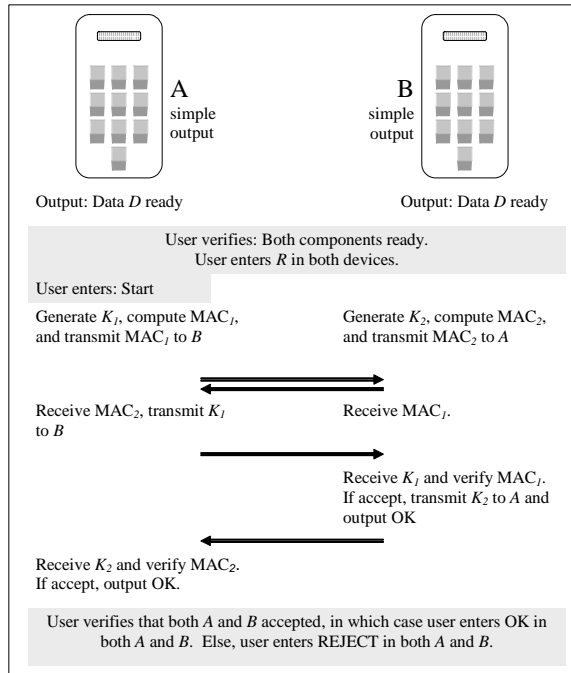


Figure 2: Manual authentication using a MAC

1. A data string D is agreed between A and B using the wireless channel.
2. The user generates a short random bit-string R , e.g. of 16–20 bits, and enters it into both devices.
3. Device A generates a random MAC key K_1 and computes the MAC value $M_1 = m_{K_1}(I_A||D||R)$, where I_A is an identifier for A and $||$ denotes concatenation of data items. Device A sends M_1 to B via the wireless link.

4. Device B generates a random MAC key K_2 and computes the MAC value $M_2 = m_{K_2}(I_B||D||R)$, where I_B is an identifier for B . Device B sends M_2 to A via the wireless link.
5. When device A receives M_2 from B (and not before), A sends B the key K_1 .
6. When device B receives M_1 from A (and not before), B sends A the key K_2 .
7. On receipt of K_2 , A uses it to recompute M_2 , where the data employed in the computation consists of its stored value of D , the expected identifier I_B , and the random value R input by the user. If the recomputed M_2 agrees with the value received from B then A indicates success.
8. On receipt of K_1 , B uses it to recompute M_1 , where the data employed in the computation consists of its stored value of D , the expected identifier I_A , and the random value R input by the user. If the recomputed M_1 agrees with the value received from A then B indicates success.
9. If (and only if) both devices indicate success, the user indicates success to both devices.

Finally note that steps 2/3 and also 4/5 may be conducted in parallel.

4.2 Analysis of the scheme

The MANA III scheme is a slightly modified version of a protocol called SHAKE [9]. Informally, the security of the scheme relies on the fact that R remains secret to the attacker (it is never sent over the air) and both A and B release a commitment (i.e. the MAC value) to the data D before releasing the key used to compute this commitment.

In order for the scheme to work, the last step must be performed, since it is indeed easy for a forger to

make a full (not successful) exchange with A , calculate R by exhaustive search, and then make a successful exchange with B . However, such an attack will be detected by the double check in the last step. Hence, the interceptor's only hope of attacking the scheme is to determine R from the MAC values, but in the absence of the keys this is infeasible.

Thus the best approach for the attacker is to guess R . The likelihood of a successful attack is thus 2^{-r} , for an r -bit value R , i.e. the odds against a successful attack are 1 in 70,000 for a 16-bit random value R , and 1 in a million for a 20-bit R .

Of course, this calculation assumes that R is chosen at random from all possible r -bit values. In practice, if R is chosen by a human, then some values will be more likely than others. This can be exploited by an attacker whose best strategy is simply to guess the most likely r -bit value. However, this is unlikely to drastically reduce the security of the system unless the attacker understands well the behaviour of the user being attacked. Also, if R is instead chosen by one of the devices, as in the variant scheme described immediately below, then the risks associated with a poor choice of R are avoided.

4.3 Variants of the scheme

Two variants of the scheme exist, both of which are included in the draft standard [6].

The first variant (originally proposed by Jakobsson [7] and listed as mechanism 3b in [6]) involves a total of r 'rounds', where r is the number of bits in R . In each round, one bit of R is used, and the devices exchange MACs and keys as in the scheme described above. While this increases significantly the amount of data exchanged between the devices, it removes the need for the user to give success indications to the devices at the end of the protocol.

The second variant (mechanism 4 in [6]) applies to the case where one device has a display and the other has a keypad. In this case, step 2 is modified so that the device with the display generates the random value R and displays it to the user, who then enters it into the other device. All other steps of the scheme remain unchanged.

5 Concluding remarks

The schemes described in this paper meet the objective of enabling two wireless devices to securely authenticate one another and agree on a shared data string. This is achieved without the need for the user to enter or compare long strings of digits. The user is typically only required to type in (or compare) around 32 binary digits (e.g. in the form of eight hexadecimal digits).

Mechanism MANA III further improves on the situation and only requires the user entry of 16 bits, although these digits must be entered into both devices or read from one device and typed into the other.

References

- [1] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, **IT-22**:644–654, 1976.
- [2] C. Gehrman and K. Nyberg. Enhancements to Bluetooth baseband security. In *Proceedings of Nordsec 2001, Copenhagen, Denmark, November 2001*.
- [3] C. Gehrman and K. Nyberg. Security in personal area networks. In C. J. Mitchell, editor, *Security for Mobility*, pages 191–230. IEE, London, 2004.

- [4] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 10118-2, Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using an n-bit block cipher*, 2nd edition, 2000.
- [5] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 10118-3: 2003, Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*, 2nd edition, 2003.
- [6] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 1st CD 9798-6, Information technology — Security techniques — Entity authentication — Part 6: Mechanisms using manual data transfer*, December 2003.
- [7] M. Jakobsson. Method and apparatus for immunizing against offline dictionary attacks. U.S. Patent Application 60/283,996. Filed on 16th April 2001.
- [8] M. Jakobsson and S. Wetzel. Security weaknesses in Bluetooth. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 176–191. Springer-Verlag, Berlin, 2001.
- [9] J.-O. Larsson. Higher layer key exchange techniques for Bluetooth security. Open Group Conference, Amsterdam, October 2001.
- [10] D. P. Maher. Secure communication method and apparatus. U.S. Patent Number 5,450,493, September 1995. Filed on 29th December 1993.
- [11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.
- [12] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons Ltd., 2002.
- [13] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols, 7th International Workshop, Cambridge, UK, April 19-21, 1999, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 172–194. Springer-Verlag, Berlin, 2000.