Location Privacy of Distance Bounding Protocols

Kasper Bonne Rasmussen
Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
kasperr@inf.ethz.ch

Srdjan Čapkun
Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
capkuns@inf.ethz.ch

ABSTRACT

Distance bounding protocols have been proposed for many security critical applications as a means of getting an upper bound on the physical distance to a communication partner. As such, distance bounding protocols are executed frequently, e.g., to keep node locations up to date, etc. We analyze distance bounding protocols in terms of their location privacy and we show that they leak information about the location and distance between communicating partners even to passive attackers. This location and distance information may be highly sensitive since it can form the basis for access control, key establishment, or be used as input to location aware applications. We analyze, in a number of scenarios, how much information distance bounding protocols leak. We further discuss several straightforward countermeasures and show why they do not provide adequate protection against distance leakage. Finally, we propose a location private distance bounding protocol that maintains the properties of existing distance bounding protocols while leaking no information about the distance measured between the communicating parties.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design —*Distributed networks, Wireless communication*; C.3 [**Computer Systems Organization**]: Special-Purpose And Application-Based Systems—*Real-time and embedded systems.*

General Terms

Security, Theory.

Keywords

Wireless Security, Distance Bounding, Information Leakage.

1. INTRODUCTION

In recent years, distance bounding protocols [4] have been proposed for several different classes of devices, e.g., wireless RF devices [27, 11], RFID [10, 8, 16], ultrasonic devices [21, 22] and UWB [14, 9]. All the proposed protocols have one thing in common: they aim to provide an efficient and accurate distance (or distance bound) between two nodes. Regardless of the type of distance bounding protocol, the distance bound is obtained from a rapid exchange of messages between two nodes called the prover and the verifier. In this paper, we analyze this rapid message exchange, common to all distance bounding protocols, in terms of the information that a passive attacker can obtain from overhearing the communication between two nodes executing a distance bounding protocol.

In general, most protocols leak some kind of information, e.g., by executing a protocol two nodes might reveal the fact that they are present within the attackers radio range. What makes the information leakage from distance bounding protocols especially severe is that the information that is leaked is the same as the nodes participating in the protocol, i.e., the prover and verifier, will obtain after the execution of the protocol. In distance bounding protocols, a passive attacker is able to deduce not only the distance between the prover and verifier, with the same accuracy as the nodes executing the protocol, but also his own position relative to the prover and verifier.

Distance bounding protocols are often used to allow nodes to build topology maps of the network or to control access to specific resources in the network, e.g., a node can only access a specific resource if it is in a specific location. In those application scenarios, the leaking of the distance will effectively give the attacker the same map of the network as the legitimate nodes have, or enable him to map out where any special access zones might be. This can cause a severe breach of security.

We analyze several straightforward ways of countering the information leakage from distance bounding protocols and we point out the weaknesses or strong assumptions appropriate for each of those solutions. We also identify eight different scenarios (different a priori attacker knowledge) that affect the amount of information leaked to the attacker. Finally, we propose a location private distance bounding protocol (LP-DB) that solves the problems outlined above. Our protocol also prevents distance leakage to an active attacker, which starts an (unauthorized) distance bounding session; we thus extend our attacker model to include active attackers.

We summarize our contributions in the following points: (i) we present a thorough investigation of the distance and location information leaked from distance bounding proto-

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

cols in various scenarios, (ii) we analyze several different straightforward countermeasures and discuss why they do not provide adequate protection against this information leakage (iii) we propose a location private distance bounding protocol that leaks a minimum amount of information.

The rest of the paper is organized as follows: Section 2 gives a quick introduction to distance bounding protocols. In Section 3 we analyse what information leaks from the distance bounding protocol and we build a basic model of the attackers knowledge. In Section 4 we analyse the possible ways of countering the leaking, and we expand the model for the attackers knowledge to take different scenarios into account. In Section 5 we describe our location private distance bounding protocol (LP-DB). In Section 6 we discuss related work and we conclude the paper in Section 7.

2. BACKGROUND

Distance bounding denotes a class of protocols in which one entity (the verifier) measures an upper-bound on its distance to another (un-trusted) entity (the prover). Distance bounding protocols were first introduced by Brands and Chaum [4] for the prevention of mafia-fraud attacks on Automatic Teller Machines (ATMs). The purpose of Brands and Chaum's distance bounding protocol was to enable the user's smart-card (verifier) to check its proximity to the legitimate ATM machine (prover). Figure 1 shows the main principle of operation of distance bounding protocols.

In distance bounding protocols, the verifier challenges the prover with a b-bit freshly generated nonce N. Upon reception of the challenge, the prover computes an (authenticated) response $f^P(N)$, and sends it to the verifier. The verifier verifies the authenticity of the reply and measures the time $t_s^V - t_r^V$ between the challenge and the response. This process is repeated k times to avoid the prover guessing (part of) N and replying before the whole challenge is received. Based on the measured time, the verifier estimates the upper-bound on the distance to the prover. The time $t_s^P - t_r^P$ between the reception of the challenge and the transmission of the response at the prover is either negligible compared to the propagation time $t_r^P - t_s^V$ or is lower bounded by the prover's processing and communication capabilities δ, i.e., $t_s^P - t_r^P \geq \delta$.

The security of a distance bounding protocol relies on the following observations. The challenge N_i cannot reach the prover before it has been sent by the verifier, and its propagation cannot be sped-up if the messages propagate at the speed of light (e.g., over a radio channel); the message propagation can therefore not be shortened by external attacks or by untrusted provers. The prover's responses are protected from external attacks by their unpredictability, and from the untrusted prover by the fact that the responses are functions $f^P(N_i)$ of the challenges; i.e., the prover cannot send back the response before receiving the challenge. Given this, distance bounding protocols provide to the verifier an upper-bound on its distance to the prover.

After the execution of a distance bounding protocol the verifier knows that the prover is within a certain distance, namely:

$$dist = \frac{t_s^V - t_r^V - \delta}{2} \cdot c$$

where δ is the processing time of the prover (ideally 0) and c is the propagation of the radio wave.

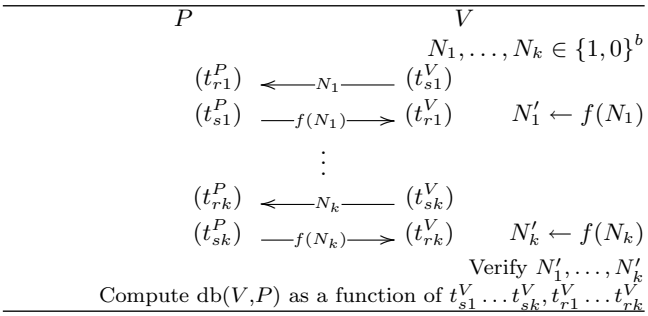


Figure 1: The main component of distance bounding protocols consists of a rapid exchange of messages where the time of flight between the prover and verifier is measured.

3. INFORMATION LEAKAGE FROM DISTANCE BOUNDING PROTOCOLS

In this section we analyze the distance information leaked from distance bounding protocols. The amount of information leaked by the execution of a distance bounding protocol depends on how much the attacker knows about his own position relative to the prover and verifier before the protocol starts. In this section we assume that the attacker has no information about the positions of the prover and verifier, except that they are in his power range. In Section 4 we will then expand the model to include the attacker's knowledge. Before we start the analysis we describe our system and attacker models.

3.1 System and Attacker Model

We consider three nodes, the prover P, the verifier V and the attacker M. The prover and verifier execute a distance bounding protocol as described in Section 2. We assume that the verifier is trusted and not compromised and that both the prover and the verifier do not deliberately give any information to the attacker.

The nature of the distance bounding protocol implies that the verifier does not trust the prover (otherwise they could use an authenticated ranging protocol [30]) but for the purpose of this analysis we assume that the prover is honest and complies with the distance bounding protocol to the best of its capabilities. The prover and the verifier are within one hop communication range and the delay introduced by the message processing of the prover δ_p and verifier δ_v are public values.

We consider that the attacker can listen to the radio communication of both the prover and the verifier. We do not require that the attacker holds any keys or any other secret material that form part of the protocol between P and V, however, the attacker does know the public parameters of the distance bounding protocol and the type of hardware used by the nodes and thus the processing time of the provers and verifiers radios.

We do not assume any kind of time synchronization between the nodes, although we do assume that the nodes can time-stamp messages with, at least, nanosecond precision; examples of such hardware can be found in [12], and hardware of more recent implementations of distance bounding and authenticated ranging protocols in [8, 16, 19].

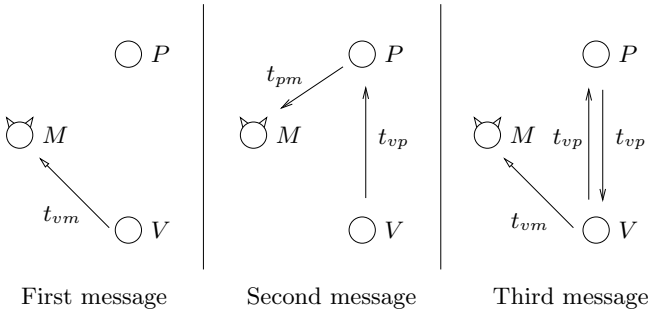


Figure 2: The paths of three consecutive messages in the rapid message exchange.

3.2 Distance leakage

If two nodes, the prover and the verifier, execute a distance bounding protocol under the assumptions described in Section 3.1 the distance between the prover and verifier leaks, even if the attacker remains completely passive, i.e., even if he does not participate in, or interfere with, the protocol execution.

In order to mount the attack the attacker needs to record the time at which the messages from the rapid message exchange phase of the distance bounding protocol arrive at his radio interface. The attacker must record the arrival time of three consecutive messages to obtain enough information to calculate the distance between P and V. The arrival times T_i of three consecutive messages are illustrated in Figure 2 and can be described by the following three equations:

$$T_0 = t_0 + t_{vm} \quad (1)$$
$$T_1 = t_0 + t_{vp} + \delta_p + t_{pm} \quad (2)$$
$$T_2 = t_0 + 2t_{vp} + \delta_p + \delta_v + t_{vm} \quad (3)$$

The attacker receives the first message at T_0 which is the time it was sent t_0 plus the time it took the signal to propagate from the verifier V to the attacker M. The next message is the prover P's response to the first message, so the time at which the attacker receives it T_1 is the sum of the time the first message was sent by the verifier t_0, of the time it took the message to propagate from V to P, of the time P took to process the message δ_p and of the time it took the message to propagate from P to M. The third message is a response to the second message and it includes two propagation times between V and P[1] $2t_{vp}$ and two processing times δ_v and δ_p.

The attacker can now find the signals time of flight between the verifier V and prover P:

$$t_{vp} = \frac{(T_2 - T_0) - \delta_p - \delta_v}{2} \quad (4)$$

In order for V and P to measure the distance between them, δ_v and δ_p must be small, and constant, public values as described in Section 2.

When the attacker has found the signals time of flight t_{vp} using (4), the distance from V to P can be found by multiplying t_{vp} by the speed of light c

$$d_{vp} = c \cdot t_{vp} \quad (5)$$

[1]This assumes that the message propagates with equal speed from V to P and from P to V, i.e., $t_{vp} = t_{pv}$

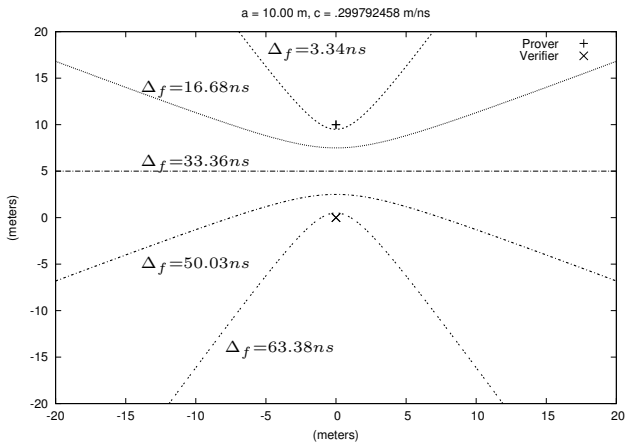


Figure 3: The hyperboles, relative to P and V, leaked by two subsequent messages of a distance bounding protocol for five different values of $\Delta_f = \Delta_1 - \delta_v$.

In order to obtain equation (3) we assume that the verifier sends its next challenge immediately after receiving the response from the prover. This is true for mutual distance bounding protocols [27, 29] but not for all types of distance bounding protocols in general. In Section 4 we show that in a number of scenarios, indeed only the two first messages are needed for the attacker to obtain the distance d_{vp}.

3.3 Location leakage

It is not only the distance between the prover and verifier that leaks. From two subsequent distance bounding messages the attacker can infer his own location (x, y) relative to the prover and verifier if he knows the distance between P and V.

For the attacker to obtain information about his position relative to the prover and verifier he needs the difference between the arrival times of two subsequent messages Δ_1. The difference between the arrival times of two subsequent messages follows from (1) and (2):

$$T_1 - T_0 = \Delta_1 = t_{vp} + \delta_p + t_{pm} - t_{vm} \quad (6)$$

If we convert from time to distance by multiplying with c on both sides, we get:

$$c\Delta_1 = d_{vp} + c\delta_p + d_{pm} - d_{vm}$$
$$c(\Delta_1 - \delta_p) - d_{vp} = \sqrt{x^2 + (d_{vp} - y)^2} - \sqrt{x^2 + y^2} \quad (7)$$

In order to describe the position of the attacker relative to the prover and verifier we have to define a coordinate system in which the prover, verifier and attacker have well defined positions. In (7) we assume the verifier is located at (0,0) and the location of the prover defines the positive direction of the y-axis, i.e., the prover is located at $(0,d_{vp})$.

To simplify the equations, we define the left side of equation (7) as a pseudo distance Δ_p:

$$\Delta_p \equiv c(\Delta_1 - \delta_p) - d_{vp} \qquad \text{for} - d_{vp} \leq \Delta_p \leq d_{vp} \quad (8)$$

which gives

$$\Delta_p = \sqrt{x^2 + (d_{vp} - y)^2} - \sqrt{x^2 + y^2}$$
$$y = \frac{\pm\Delta_p\sqrt{4x^2 + d_{vp}^2 - \Delta_p^2} + d_{vp}\sqrt{d_{vp}^2 - \Delta_p^2}}{2\sqrt{d_{vp}^2 - \Delta_p^2}} \quad (9)$$

Equation (9) describes a hyperbole relative to the prover and verifier, on which the attacker must be located. Different examples of such hyperboles can be seen in Figure 3 along with the corresponding $\Delta_f = \Delta_1 - \delta_p$ values. In essence, the messages sent by the prover and verifier work like beacons in a TDOA [33, 7] system, only here the beacons are not transmitted at the same time, but in a rapid sequence.

3.4 Attacker initiates the Distance Bounding Protocol

Another way information can leak from distance bounding protocols is if the attacker takes the role of the prover (or verifier) and initiates a distance bounding session with the other node. This is a deviation from the passive attacker model since the attacker is now actively sending bits to force the distance to leak.

Most distance bounding protocols do not have any form of authentication until after the rapid exchange of messages [4, 30] so even if the attacker does not hold a valid key he can initiate the protocol and trick the prover (or the verifier) into completing the rapid message exchange, at which point the attacker will know the distance to the prover (verifier) and abort the protocol before completing the authentication.

To the best of our knowledge no existing distance bounding protocol include authentication in the setup phase. Some use a shared key to communicate before the ranging phase begins [10, 26] but do not prevent an external attacker from initiating the protocol and completing (part of) the ranging phase. Even if authentication is included in the setup phase of the distance bounding protocol, the individual messages of the rapid message exchange are not authenticated so an attacker can still wait for two nodes to initiate the protocol and then take over the rapid message exchange by overshadowing the signal from the valid node.

4. INFORMATION LEAKAGE COUNTERMEASURES

In order to prevent information leakage from distance bounding protocols, the attacker must be prevented from calculating the time of flight of the signal between the verifier and the prover, as shown in equation (4). In this section, we explore various solutions to this problem and we show why each of the solutions fails to provide full protection against distance and location leakage attacks. We will use the lessons learned in this section to construct a location private protocol (Section 5).

The protocols presented in this section all use single bits as the messages $N_1, \ldots, N_k$ and the function applied by the prover is $\oplus$ (xor). The protocols that represent different countermeasures, all have weaknesses that limit their effectiveness but they highlight why the problem of information leakage is not trivial to solve.

4.1 Adding random delay between messages

One way to make the calculation of the time of flight of the signal between the prover and verifier t_{vp} harder for the attacker, is by adding a random delay between the messages of the rapid message (rapid bit) exchange in the protocol. In this case, a new protocol is needed to make sure that the prover and verifier can still compute the correct distance. Such a protocol (Figure 5) will be discussed at the end of this section.

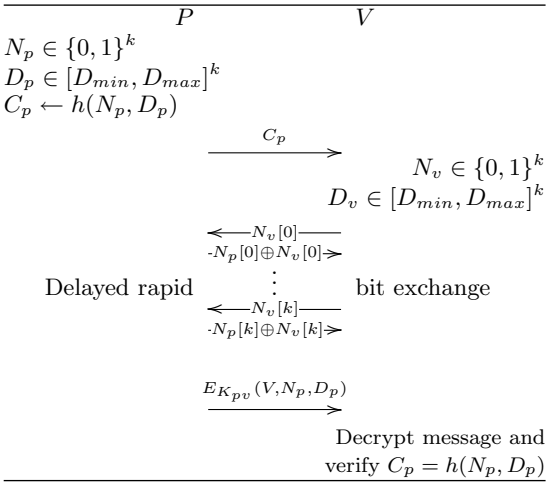


Figure 5: Distance bounding protocol with random delay between messages.

If the prover and verifier add a random delay before sending each bit of the rapid bit exchange, the equations describing the arrival time of the three subsequent messages will be modified to include the random delay for both the prover and the verifier:

$$T_0 = t_0 + t_{vm} \quad (10)$$
$$T_1 = t_0 + t_{vp} + \delta_p + \Omega_p + t_{pm} \quad (11)$$
$$T_2 = t_0 + 2t_{vp} + \delta_p + \Omega_p + \delta_v + \Omega_v + t_{vm} \quad (12)$$

where Ω_p is a random delay added by the prover P and Ω_v is a random delay added by the verifier V. The calculation of the time of flight now becomes:

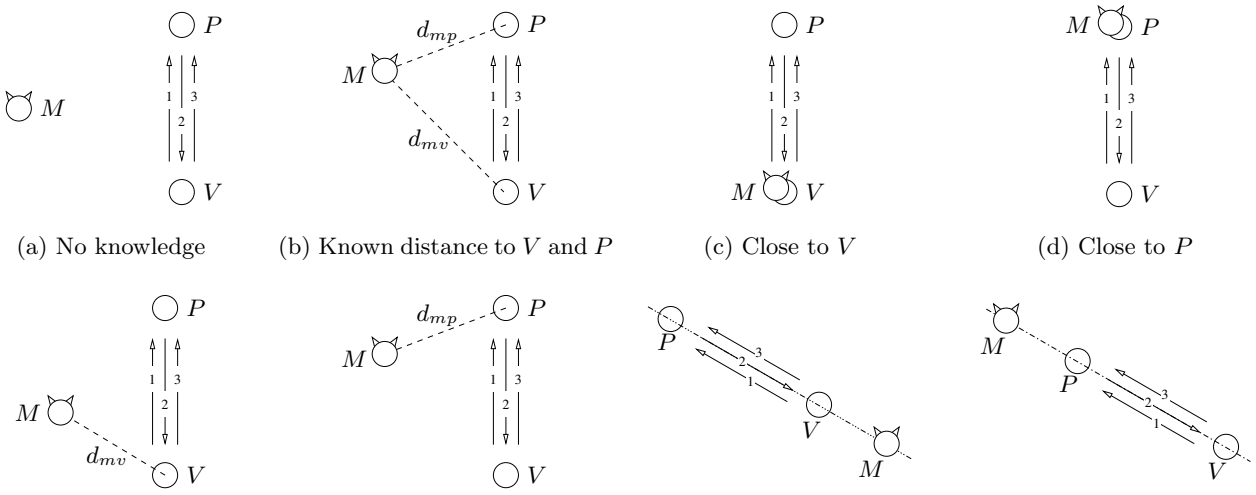
$$t_{vp} = \frac{(T_2 - T_0) - \delta_p - \Omega_p - \delta_v - \Omega_v}{2} \quad (13)$$

If the random delays Ω_p and Ω_v are set to 0 this equation is the same as equation (4).

Equations (10) – (12) will vary depending on the attackers knowledge about his own position relative to the prover and verifier, e.g., if the attacker knows his distance to the prover and verifier he will not need the third message to calculate t_{vp}. We have identified eight different scenarios, shown in Figure 4, that represent different attacker knowledge. In the rest of this paper we will refer to these scenarios by the letter used in Figure 4, e.g., the scenario where the attacker has no knowledge of the distance to either P or V is referred to as scenario (a). Table 1 contains the equations for message arrival time in all eight scenarios.

The equations describing the message arrival times are derived as described in Section 3.2. The '-' in the third column of Table 1 means that no equation is needed for T_2 in these scenarios because the first two equations are sufficient for the attacker to compute the distance between the prover and verifier.

If we look at the equations for t_{vp} in scenario (b), (c) and



(a) No knowledge (b) Known distance to V and P (c) Close to V (d) Close to P

(e) Known distance to V (f) Known distance to P (g) Directly behind V (h) Directly behind P

Figure 4: Scenarios represent the attacker's knowledge about his position relative to V and P. In all eight scenarios the verifier V and the prover P are executing a distance bounding protocol and the attacker M is able to listen to the traffic between V and P.

Scenario	Reception time of the first message T_0	Reception time of the second message T_1	Reception time of the third message T_2	Time of flight between P and V t_{vp}	
(a, e, f)	t_0+t_{vm}	$t_0+t_{vp}+\delta_p+\Omega_p+t_{pm}$	$t_0+2t_{vp}+\delta_p+\Omega_p+\delta_v+\Omega_v+t_{vm}$	$((T_2-T_0)-\delta_p-\Omega_p-\delta_v-\Omega_v)/2$	
(b)	t_0+t_{vm}	$t_0+t_{vp}+\delta_p+\Omega_p+t_{pm}$	-	$(T_1-T_0)-\delta_p-\Omega_p-(t_{pm}-t_{vm})$	
(c)	t_0	$t_0+2t_{vp}+\delta_p+\Omega_p$	-	$((T_1-T_0)-\delta_p-\Omega_p)/2$	
(d)	t_0+t_{vp}	$t_0+t_{vp}+\delta_p+\Omega_p$	$t_0+3t_{vp}+\delta_p+\Omega_p+\delta_v+\Omega_v$	$((T_2-T_0)-\delta_p-\Omega_p-\delta_v-\Omega_v)/2$	(*)
(g)	t_0+t_{vm}	$t_0+2t_{vp}+\delta_p+\Omega_p+t_{vm}$	-	$((T_2-T_0)-\delta_p-\Omega_p)/2$	
(h)	$t_0+t_{vp}+t_{pm}$	$t_0+t_{vp}+\delta_p+\Omega_p+t_{pm}$	$t_0+3t_{vp}+\delta_p+\Omega_p+\delta_v+\Omega_v+t_{pm}$	$((T_2-T_0)-\delta_p-\Omega_p-\delta_v-\Omega_v)/2$	(*)

(*) further more we have that $(T_1 - T_0) = \delta_p + \Omega_p$

Table 1: Message reception times T_0, T_1 and T_2 and the resulting equations for the time-of-flight between the verifier and prover t_{vp} for the eight scenarios described in Figure 4.

(g) (fourth column in Table 1), only the random delay added by the prover P has an effect on the attacker's calculations. Similarly in scenario (d) and (h) only the random time added by the verifier V has an effect on the calculations since we have $(T_1 - T_0) = \delta_p + \Omega_p$.

This means that in order to provide effective countermeasures (using the random delay based protocol) in all scenarios, i.e., regardless of the a priori knowledge of the attacker, both the prover and the verifier must add a random delay between the messages.

In Figure 5 we give an example of a protocol where both the prover and the verifier adds a random delay between the messages in the rapid bit exchange. In this protocol the prover first selects a k bit nonce N_p and a delay vector D_p with values between D_{min} and D_{max}. The prover then creates a commitment by hashing the two values and sends the commitment to initiate the protocol. The verifier picks his own nonce N_v and his own delay vector D_v and starts the *delayed rapid bit exchange* phase. The delayed rapid bit exchange phase takes k rounds (one round for each bit in the nonces). In the ith round the verifier will wait for $D_v[i]$ nanoseconds and then send the challenge. When the prover receives the challenge he will wait for $D_p[i]$ nanoseconds before sending back the response. After k rounds the entire nonce has been exchanged and the prover opens the commitment made in the setup phase so the verifier can check if the nonce that was exchanged in the delayed rapid bit exchange phase is correct and subtract the random delays D_p from the round-trip time of each round. Once this step is done the verifier has all the information needed to reconstruct the time-of-flight of each bit and estimate the distance to the prover.

The problem with such a protocol is that the prover also needs to add a delay. If the prover is allowed to add a random delay to the messages, the verifier can not be sure that the prover actually waits as long as it claims. Even if the prover commits to a series of delays in the setup phase of the protocol the prover could still cheat consistently on all delays, by subtracting a fixed amount from all the delay values in his delay vector, thus making himself appear closer to the verifier than he actually is. Since the same delay must be subtracted from all the values in the provers delay vector (in order to preserve consistency), this problem could be solved by requiring that at least one of the delay values is 0 but

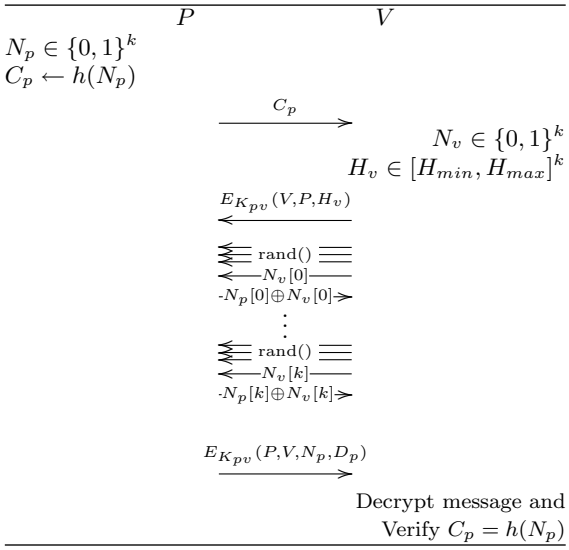


Figure 6: Distance bounding protocol with multiple challenges.

in this case the attacker would learn the same information as without any delays at all, just by looking at the fastest response.

Because this method would give the prover the ability to claim a false location closer to the verifier, thus destroying one of the most important properties of the distance bounding primitive, this method can not be used as an effective countermeasure.

4.2 Sending multiple challenges

Another way to add randomness to the attackers calculation is for the verifier to send multiple challenges to the prover before the prover responds (correctly) to one of them. This only works if the attacker is unable to distinguish between a transmission from the prover and a transmission from the verifier, otherwise the attacker will just wait for a response from the prover and then assume that it was a response to the last message sent by the verifier.

If multiple challenges are sent, a protocol is needed to make sure that the prover and verifier agree to which challenge the prover must (correctly) reply. We present such a protocol in Figure 6.

In the distance bounding protocol in Figure 6, the prover first selects a k bit nonce N_p and transmits a hash of that nonce to the verifier as a commitment. The verifier then picks his own nonce and generates an array of values that represent the number of messages the prover must receive before answering. This array is then sent to the prover. After this initial exchange of messages the rapid bit exchange starts and the verifier starts sending challenges to the prover. When the required number of challenges for round i has been received, the prover responds with the challenge xor'ed with the ith bit of his own nonce.

The verifier must send out the challenges quickly enough (or close enough together) that a potential attacker can not distinguish the provers responses from the verifiers challenges based on the inter message timing.

An attacker listening to a distance bounding protocol where multiple challenges are sent, will have to guess which one is the right challenge. When the attacker has guessed the challenge, he can assume that the next message is the response. If the prover sends a random number of challenges (between 1 and n) for each response, the probability that the attacker can guess m challenges in a row is:

$$P(attacker\_success) = \left(\frac{1}{n}\right)^m \tag{14}$$

For $n = 10$ challenges and $m = 3$ consecutive messages, the probability of the attacker successfully guessing the three correct challenges is $(1/10)^3 = .001$. Note that $m = 3$ implies scenario (a), (d), (e), (f) or (h) since, only in these scenarios, the attacker needs three messages to learn the distance between V and P.

A sophisticated attacker can use, e.g., signal fingerprinting [18] or received signal strength (rss) [31] to distinguish transmissions from the prover and verifier with a certain probability, so the assumption of indistinguishability of the prover and verifiers signal is very strong.

The relatively high probability of successfully guessing three challenges and the assumption that the attacker can not distinguish between transmissions from the prover and verifier make this protocol an inadequate countermeasure to information leakage.

4.3 Send challenges with a fixed interval

If the verifier sends challenges with a fixed interval, say, every 100ms, the attacker will not be able to derive any information from the time between receiving the response from the prover T_1 and receiving the next challenge from the verifier T_2. Essentially the attacker would only get T_0 and T_1.

This technique effectively prevents the distance and relative position from leaking in scenarios (a, d, e, f, h) but not in (b, c, g) since only two messages are needed to calculate the distance in these scenarios. So while sending challenges with a fixed interval is a strong, and easy to implement, countermeasure it does not solve the problem completely.

4.4 Hiding the transmission of messages

A different approach to counter information leakage from distance bounding protocols is to hide the fact that any messages are being sent at all, thus preventing the attacker from obtaining T_0, T_1 and T_2.

In this section we will look at the advantages and disadvantages of using direct sequence spread spectrum (DSSS) [17] or frequency hopping (FH) [13] to make detection of the transmission harder for the attacker. In this context, the immediate goal of using any spreading technique is to deprive the attacker of information regarding the arrival time of the messages in the rapid bit exchange. Both DSSS and FH have a number of features that become problematic when nanosecond accuracy is needed [24]. To better illustrate the impact of spreading and de-spreading on the delay, we give a short review of the receiver synchronization procedure.

4.4.1 DSSS receiver synchronization

Direct sequence spread spectrum (DSSS) works by modulating the original data signal with a high frequency chip-signal, or spreading code, thus spreading the resulting DSSS signal over a wider frequency band. If the transmitter uses the same amount of power to transmit the DSSS signal as would have been used to transmit the original signal, less

energy is transmitted on each frequency (since more frequencies are used). The DSSS signal can become difficult to separate from channel background noise for someone who does not know the correct chip-sequence [24, 5].

In order to receive a DSSS signal the sender and receiver radios must be tightly synchronized[2] such that the spreading code and the de-spreading code are applied correctly. The process of synchronizing the receiver radio to the sender radio is called signal accusation and is performed each time a new transmission begins. The receiver must be tuned to exactly match the phase of the incoming signal to correctly apply the de-spreading code [24].

Acquisition is accomplished in a two step process. First a rough synchronization is performed to synchronize the receiver to within one chip of the incoming signal, then a phase locked loop (PLL) takes over and performs the final fine grained synchronization. The PLL will track the phase of the incoming signal and compensate for any Doppler shifts or other changes (e.g., frequency drift of the two radios), so once the signal is acquired the de-spreading can be performed with negligible delay.

The problem with this process is that in the rapid bit exchange each message (i.e., each bit) is a separate transmission and therefore the signal must be acquired every time a new bit arrives.

The acquisition delay depends on the design of the receiver. We briefly describe two receivers, one with a parallel design and one with a serial design. For a detailed description of how these two receiver designs work, please refer to [24, p. 746], here we will just describe the delay introduced by the two designs.

A parallel receiver design with $2N_c$ signal correlations can synchronize a receiver if it is within N_c chips of correct alignment. In such a case the signal acquisition time is given by the following equation:

$$T_{acq} = \lambda T_c \tag{15}$$

where λ is the number of chips that must be received before a decision is made and T_c is the time it takes to receive one chip. The equation shows that the more chips the receiver receives before a synchronization decision is made, the longer it takes to acquire the signal but a bigger λ also means that the probability of erroneous acquisition is reduced.

The popular and much simpler and cheaper serial receiver design has the disadvantage of a higher (and more unpredictable) acquisition time. In the following equations P_D is defined as the probability of correct correlation and P_{FA} as the probability of false alarm as given by [24]:

$$(T_{acq})_{max} = 2N_c\lambda T_c \tag{16}$$

$$(T_{acq})_{avg} = \frac{(2-P_D)(1+KP_{FA})}{P_D}(N_c\lambda T_c) \tag{17}$$

$$\sigma_{acq} = (2N_c\lambda T_c)^2(1+KP_{FA})\left(\frac{1}{12}+\frac{1}{P_D^2}-\frac{1}{P_D}\right) \tag{18}$$

It should be noted that other synchronization techniques with faster acquisition times do exist, e.g., RASE [32], however these techniques are often highly susceptible to noise and interference [24] and are thus not well suited for our purpose.

[2]Synchronization in this case does not refer to time synchronization on the application layer, but radio synchronization in order to correctly apply the spreading code.

After the rough synchronization, control is handed off to the phase locked loop (PLL) which will need a few more chips to acquire complete lock.

It is clear that the serial receiver design introduces more random delay than the parallel design, but even the parallel design can fail to acquire the signal (if the synchronization error is bigger than N_c chips). If that happens the receiver must retry the acquisition and that will introduce uncertainty in the delay. Since any random delay introduced by the prover will be impossible for the verifier to compensate for, it will adversely affect the result of a distance bounding protocol.

4.4.2 DSSS code and message length

There are also several problems relating to the length of the chip sequence (or spreading code) used.

By the length of the spreading code we mean the number of chips of code per bit of data. The longer the spreading code is, the harder it is for the attacker to detect the transmission but it also takes longer to send and receive a single bit of data (if the rate is fixed). This has implications on the granularity of the distance bound since the prover must wait for the entire chip sequence to arrive before responding, in order to verify that it is indeed the right spreading code being used.

Another problem with the length of the message is that in order to acquire the signal, i.e., perform the course synchronization, the receiver must get at least λ chips. If the message is only one bit long then a preamble must be used. This further increases the overall transmission time for each message and thus reduces the granularity of the distance bound.

4.4.3 Off-line attacks

A problem that signal spreading does not solve is off-line attacks. A determined attacker can record the signal in the entire band used for transmission and then do a brute force attack on the spreading codes. Since the timing is not obscured by the spreading technique (beyond what happens during signal acquisition) the attacker will get the same information as with an online attack. A brute force attack might be infeasible for DSSS but for FH it is trivial.

Because of the problems outlined in this section, signal spreading alone can not be used to effectively prevent information leaking without introducing other problems but we will show that it can be an additional protection mechanism when used in conjunction with our location private distance bounding protocol described in Section 5.

5. LOCATION PRIVATE DISTANCE BOUNDING PROTOCOL

Distance bounding protocols used to build topology maps of a network or to control access to specific resources in a network are prime targets for attack. The leaking of information from these protocols will give the attacker the same map of the network as the legitimate nodes have, or enable him to map out where any special access zones might be. This can be a severe breach of security.

In this section we present a distance bounding protocol designed to minimize the amount of information that leaks to an attacker. In order to prevent the attacker from being

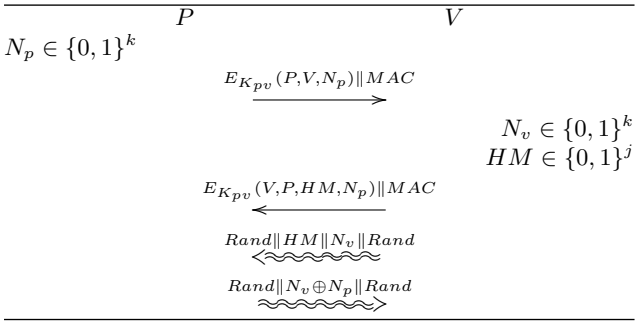


Figure 7: Distance bounding protocol that does not leak information to a passive or active attacker.

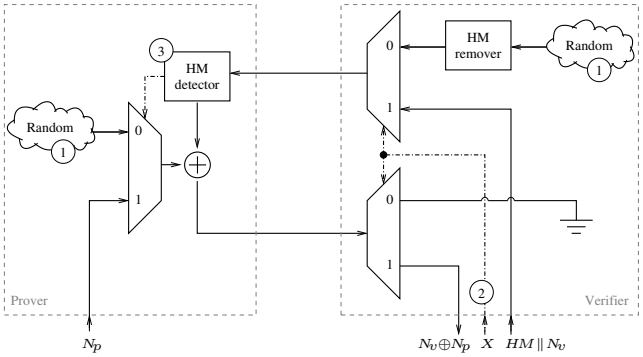


Figure 8: A functional diagram of the prover and verifiers radios.

able to learn the transmission times of the rapid bit exchange our location private distance bounding protocol is based on streams as illustrated in Figure 7.

Our protocol works in the following way: The prover picks a k bit nonce and sends it to the verifier, encrypted with a shared key. The verifier also picks a k bit nonce and a hidden marker *HM* that will mark the beginning of valid data in the data stream. The hidden marker is sent encrypted to the prover. The verifier then starts sending random data to the prover. The prover will xor this data with his own stream of random bits and send it back to the verifier.

The way in which the streams are started and stopped is important in order not to leak any information. The following describes the timing of the various steps after both streams are established:

After both streams are established, the prover will continuously monitor the stream for the hidden marker while xor'ing it with random bits and sending it back. When the last bit of the hidden marker appears the prover will start xor'ing the incoming data (which should now be the verifiers nonce) with his own nonce and transmit this back to the verifier. The time at which the verifier starts transmitting the hidden marker *HM* followed by the nonce N_v is randomly chosen (within the setup window w_s) so an attacker can not deduce the transmission time of the nonce from the transmission time of the start of the stream. The setup window only begins after both streams are established.

The moment the verifier has sent the last bit of the hidden marker he will begin to save the bit stream from the prover. This will enable the verifier to count the number of bits between the transmission of N_v and the reception of $N_v \oplus N_p$.

After the final bit of the verifiers nonce N_v has been sent, the verifier continues to transmit random bits to make it harder for the attacker to determine when the actual transmission ended. The prover also switches back to random bits after sending $N_v \oplus N_p$. When $N_v \oplus N_p$ has been successfully received, the verifier will stop the continuous transmission after a short random delay, and after a another random delay the prover will do the same. The short random delay must be there or an attacker can measure the time between the end of the verifiers transmission and the end of the provers transmission to estimate the time of flight of the signal.

Once both streams have been terminated the prover will count the number of bits in the saved bit stream between the moment the verifier started to send out N_v and the moment the first bit of $N_p \oplus N_v$ came back and subtract the known processing time of the provers radio. This gives the verifier a round trip time and thus an upper bound on the distance to the prover.

The location private distance bounding protocol works at any bit rate, but for it to be useful, the bit rate must be high enough to give a reasonable granularity. A bit rate of 1Gb/s will give a granularity of about 30cm. We will look at the bit rate requirements of the protocol in more detail in Section 5.5.

5.1 External attackers and malicious provers

A malicious prover can not cheat on the distance measurement for the same reason that a prover using an existing distance bounding protocol described in Section 2 can not cheat; he does not know the value of the challenge bit $N_v[i]$ before he has to send the response. The bit rate of the streams must be the same in both directions, i.e., from the verifier to the prover and back. That means that the prover can only send the ith bit of $N_v \oplus N_p$ back after he receives the ith bit of N_v. Even if the prover tries to guess the next bit he has to wait until it is time to send the next bit before doing so (otherwise he will violate the bit rate) and by that time, he knows the actual value of the next bit of N_v, so there is no opportunity to guess.

An external attacker can not get the distance to either the prover or the verifier by initiating the protocol as it requires communication using a shared key before the streams are started. Once N_p and the *HM* have been exchanged, the attacker can not learn anything from listening to the streams since the data transmitted looks completely random and because the streams are one long continues transmission the attacker can not deduce any message arrival times. We analyse the attackers chance of guessing valuable information in Section 5.4.

5.2 Construction of the radios

To make it clear how location private distance bounding protocol handles streams, we give a short description of a possible construction of such a system.

A functional diagram of the prover and verifier is given in Figure 8. The diagram shows in a schematic way how the streaming part of the protocol could be realized. To read the diagram assume that all the multiplexers start in position 0. The prover sends out bits from a (pseudo) random source while making sure that the *HM*-sequence does not accidentally occur in the random bit stream.

When the bit stream reaches the prover it is fed through an *HM*-detector that has two outputs. The first output

is the unchanged stream itself and the second output is a signal that makes the prover's multiplexer switch from input 0 to input 1 if the *HM*-sequence is found. The unchanged stream is xor'ed with random bits (when the multiplexer is at position 0) and then sent back to the prover.

When the verifiers multiplexers are switched to the state marked 1 he starts the transmission of the hidden marker *HM* and the nonce N_v. When the *HM*-sequence reaches the prover he will detect it and switch his own multiplexer thus sending back $N_p \oplus N_v$.

5.3 HM remover and HM detector

In order to make sure that the random bits used in the beginning and end of the location private distance bounding protocol does not accidentally contain an *HM*-sequence, the verifier must have a mechanism to ensure that if such a sequence does occur it will be altered. The way in which the random bit stream is altered is important as the prover might introduce a bias in the bit stream and make the output easily distinguishable from pure random.

In the following we assume that our *HM*-remover is a shift-register with random bits coming in from the right and going out through the left side (like in Figure 8). The *HM*-remover has enough positions in its buffer to hold the size of the *HM*-sequence N and all N bits are inspected in parallel. When the left most bit is sent out a new random bit comes in from the right, shifting the bits in the *HM*-remover. Only the bits currently in the *HM*-remover can be altered and the *HM*-remover is stateless, i.e., it does not "remember" what bits it sent out.

The naive way to alter the stream is to flip a random bit if the buffer matches the *HM*-sequence. However while doing so will make sure that the bit string in the buffer is no longer the *HM*-sequence, flipping a random bit might accidentally recreate the *HM*-sequence further ahead in the bit stream, e.g., suppose the *HM* is '1 1 0 0' and the bit stream is '... 1 1 0 1 1 0 0'. The four last bits in this stream make up the *HM*-sequence and needs to be changed, but if the verifier flips bit number four from the right (randomly chosen) it will create the *HM*-sequence out of the first four bits.

One way the *HM*-remover can avoid accidentally creating the *HM*-sequence further ahead in the bit stream is by always flipping the last (left most) bit in the buffer if the buffer matches the *HM*-sequence, but that will introduce a bias since the last bit of the *HM* stays the same through out a session and flipping it would create a slight overweight of either ones or zeros.

To avoid this situation we propose the following way of modifying the bit stream: When creating the *HM* make sure that the last two bits are different, then, whenever an unintentional *HM*-sequence is detected by the verifier, the two least significant bits are flipped, i.e., '0 1' → '1 0' or '1 0' → '0 1'. If the two least significant bits are different, flipping them will not introduce a bias in the stream of random bits since the number of ones and zeros will remain constant and it will ensure that the buffer no longer matches the *HM*-sequence.

What about unintentionally creating the *HM*-sequence from previously processed output and the newly flipped bits? We prove that this situation can only occur in two situations, namely if the *HM*-sequence consists of all ones or all zeros (except the last bit which must be different). By eliminating these two extreme possibilities we make sure that flipping the two least significant bits will never create the *HM*-sequence.

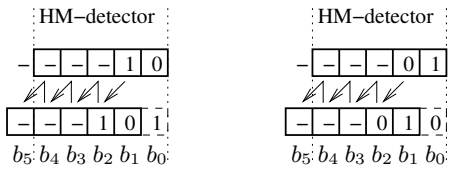


Figure 9: The *HM*-sequence ends in '1 0' (left) or '0 1' (right) so if the flip of the two least significant bits will cause the sequence to be recreated, if must be because the whole sequence was ones (left) or zeros (right).

To prove that only a sequence of all zeros (or all ones) can create the *HM*-sequence if the last two bits are flipped, we use the two illustrations in Figure 9. We start with no assumptions on the bit sequence other than it ends in '1 0' or '0 1'. The two sub figures in Figure 9 illustrate the situation where the *HM*-sequence ends in '1 0' and '0 1' respectively. We will just describe the '1 0' (left) case as the explanation applies to both situations.

The top row depicts six bits of the pseudo random bit stream that passes through the *HM*-detector. Now suppose the bits b_0–b_4 matches the *HM*-sequence that must mean that the last two bits are '1 0'. The second line depicts the situation when the last two bits are flipped. Now we see that, first of all, if a new valid *HM*-sequence is created it must mean that b_2 must be one, otherwise the bits would not create a valid *HM*. That in turn requires all the rest of the bits to be one in order to create a valid *HM*-sequence. □

That means that for the verifier to generate a 160 bit *HM*-sequence he should generate a 159 bit random nonce and then append a one or a zero such that the final two bits of the *HM*-sequence are either '0 1' or '1 0'. The only restriction that applies to the random nonce is that it can not be all zeros or all ones.

5.4 Attackers guessing space

Transmitting a stream prevents the attacker from obtaining information about when the verifier sent the nonce to the prover. As previously explained, the verifier will establish the stream and then send random meaningless data for a while before sending the hidden marker and the nonce, the amount of time is chosen at random within the setup window w_s. The attacker has two options when trying to guess the start of the nonce. The attacker can either guess the *HM*-sequence with a probability of 2^{j-1} where j is the length of the *HM* (the last bit is known once $j-1$ bits have been guessed), e.g., 2^{159} for a 160 bit *HM*, or he can try to guess the start of the *HM*-sequence without actually guessing the sequence.

For a bit rate of 1 Gb/s and a setup window of 500ms there are $5 \cdot 10^8 \approx 2^{30}$ possible starting points for the *HM*-sequence. However since the *HM* has to end in either '0 1' or '1 0' the attacker can rule out starting points that do not have one of these combinations at the end (i.e., end in '1 1' or '0 0'), assuming the attacker knows the length of the *HM*. That gives a guessing space for the attacker of $2.5 \cdot 10^8 \approx 2^{29}$ possible start locations for the *HM*. Given that the attacker only has one guess we believe that to be sufficient. If a larger guessing space is needed the prover can

either increase the setup window w_s or the protocol can be executed with a higher stream bit rate.

The reason the attacker only has one guess is that, although he can guess as many times as he wants, he has no way of verifying the guess, even if he happens to guess correctly, so if he wants to attack the protocol he will have to make a single guess and hope it is correct.

5.5 Bit rate and modulation

The verifier measures the time it takes for the prover to respond to a challenge by counting the number of bits that separates the start of his own transmission of N_v and the reception of $N_p \oplus N_v$. This is the only way to measure time since the continuous stream is already established by the time the prover must send back $N_p \oplus N_v$, and therefore the prover can only add data to the stream and not send asynchronous messages.

As a consequence of this, the time granularity with which the prover can respond is equal to the bit rate. This means that the bit rate must be sufficiently high to give a good granularity of the distance. If the bit rate is 1Gb/s, i.e., one bit every nanosecond, the distance granularity is approximately 30cm. Bit rates of 1+ Gb/s are not yet commonly used in consumer electronics but several interesting products are already on the market (early 2008). WUSB [2] have achieved a bit rate of up to 480 Mbit/s giving a distance granularity of about 60cm. There is also Sony's TransferJet [3] with bit rates up to 560Mbit/s in optimal conditions (about 50cm) and the wireless network standard IEEE 802.11n [6] offers speeds of up to 248 Mbit/s (in theory) giving a granularity of about 1.15 meters.

IEEE 802.15.3-2003 [25] is a MAC and physical layer standard for high-rate (1100 to 1055 Mb/s) wireless personal area networks (WPANs) which gives a distance granularity over shorter distances of 27 – 28 cm.

It is not only the bit rate that is important for the protocol but also the modulation. Suppose a bit rate of 1 Gb/s is achieved with QPSK [1] modulation. That does not mean that the radio receives one bit every nanosecond but rather it receives two bits every two nanoseconds, reducing the distance granularity to 60 cm.

5.6 Hiding the stream with DSSS

We saw in Section 4.4 that DSSS can not be used alone as a protection mechanism due to the unpredictable acquisition time. This unpredictability is not a problem in the stream based protocol however, since the exact arrival time of the first bits of the stream have no meaning, and as noted in Section 4.4, once the connection is established and the verifier and prover are synchronized, the delay introduced by the DSSS system is negligible.

For the prover and verifier to use DSSS the verifier must pick a spreading code at the beginning of the protocol and send this code encrypted to the prover along with the *HM*-sequence. Following that, the stream is established as explained, with the exception that it is now being transmitted using DSSS with the spreading code chosen by the verifier.

Using DSSS will force the attacker to search for the right spreading code before trying to guess the position of the *HM* within the stream. If the signal is spread over a large frequency band the attacker might not even be able to separate it from channel background noise. In this case the use of a hidden marker in the stream is superfluous.

Another benefit of using DSSS on top of the stream based distance bounding protocol is that several nodes can execute the protocol simultaneously with minimal interference, thus making the system more robust. If DSSS is not used, the protocol will occupy the channel for the duration of the stream.

The only downside, and the reason that DSSS is only mentioned as an extension to the protocol, is that it either increases the (already high) bit rate of the stream or it decreases the granularity of the distance bound. This is a technical limitation that may become less important as new high bit rate UWB systems become more common.

6. RELATED WORK

Distance bounding was first introduced by Brands and Chaum [4] in 1993. Since then the notion has been used in a number of different applications for wireless networks including [23, 27, 30] and in sensor networks [15, 29, 28, 22].

Distance bounding protocols have also been proposed in other contexts, e.g., for RFIDs [10, 8, 16] and ultra wide band (UWB) devices [14, 9].

The information leaked by distance bounding protocols has received very little attention so far. To the best of our knowledge the only other attempt to solve some of the problems analyzed in this paper is a patent by Sahinoglu, Orlik and Molisch [20] in which the authors present a protocol that provide increased privacy for the prover.

The protocol in [20] is similar to the protocol we analyze in Section 4 in that the prover and verifier agree on a delay before the actual ranging phase begins. The solution does not consider the case where the prover is malicious and tries to cheat on the distance, but only attempts to provide an honest prover with increased privacy. The privacy is achieved using a fixed set of autocorrelation codes from which one is chosen at random by the verifier when the protocol begins.

While the solution presented in [20] does go some way towards addressing the problem of information leakage, its scope is significantly different from ours. It targets prover privacy from external attackers only and the solution does not provide protection against attacks in which the attacker tries all autocorrelation codes off line as described in Section 4.4.3.

Our work is the first to do a comprehensive analysis of the information that leaks from a distance bounding protocol and to propose a solution that protects against both malicious provers, passive eavesdroppers (including off-line attacks) and attackers that try to actively initiate a distance bounding session.

7. CONCLUSION

Distance bounding protocols for determining an upper bound on the distance between two entities have been proposed for many security critical applications including secure localization and access control.

In this work, we showed that existing distance bounding protocols leak distance and location information to an attacker who overhears the communication between the prover and verifier. We identified eight scenarios that represents different a priori knowledge of the attacker. In each scenario we analyzed how much information is leaked and how the attacker can reconstruct the distance between the prover and verifier. We also showed how the attacker can find his

own position relative to the prover and verifier. We further analyzed different straightforward countermeasures and discuss why they do not provide adequate protection against information leakage in distance bounding protocols. Finally, we presented a location private distance bounding protocol that preserves all the attractive properties of other distance bounding protocols, but does not leak any range and distance information to the attacker. Our protocol can thus be used for location private distance bounding or as a basis for location private secure localization.

8. REFERENCES

[1] QPSK modulation demystified. http://www.maxim-ic.com/appnotes.cfm/appnote_number/686/, 2000.

[2] Certified wireless usb from the usb-if. http://www.usb.org/developers/wusb/, 2008.

[3] Sony develops new close proximity wireless transfer technology "TransferJet". http://www.sony.net/SonyInfo/News/Press/200801/08-002E/index.html, 2008.

[4] S. Brands and D. Chaum. Distance-bounding protocols. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 344–359. Springer-Verlag New York, Inc., 1994.

[5] Liang Chang, Fuping Wang, and Zanji Wang. Detection of dsss signal in non-cooperative communications. *Communication Technology, 2006. ICCT '06. International Conference on*, pages 1–4, Nov. 2006.

[6] Broadcom Corporation. 802.11n: Next-generation wireless lan technology. http://www.broadcom.com/docs/WLAN/802_11n-WP100-R.pdf, 2006.

[7] A. Dersan and Y. Tanik. Passive radar localization by time difference of arrival. *MILCOM 2002. Proceedings*, 2:1251–1257 vol.2, 7-10 Oct. 2002.

[8] Saar Drimer and Steven J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proceedings of the USENIX Security Symposium 2007*, 2007.

[9] S. Gezici, Zhi Tian, G.B. Giannakis, H. Kobayashi, A.F. Molisch, H.V. Poor, and Z. Sahinoglu. Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks. *Signal Processing Magazine, IEEE*, 22(4):70–84, July 2005.

[10] Gerhard P. Hancke and Markus G. Kuhn. An rfid distance bounding protocol. In *SecureComm '05: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 67–73, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks. In *Proceedings of the IEEE Conference on Computer Communications (InfoCom)*, San Francisco, USA, April 2003.

[12] Multispectral Solutions Inc. UPS (urban positioning system). http://www.multispectral.com.

[13] Z. Kostic and I. Maric. Dynamic frequency hopping in wireless cellular systems-simulations of full-replacement and reduced-overhead methods. *Vehicular Technology Conference, 1999 IEEE 49th*, 2:914–918 vol.2, Jul 1999.

[14] J.-Y. Lee and R.A. Scholtz. Ranging in a Dense Multipath Environment Using an UWB Radio Link. *IEEE Journal on Selected Areas in Communications*, 20(9), December 2002.

[15] Catherine Meadows, Paul Syverson, and LiWu Chang. Towards more efficient distance bounding protocols for use in sensor networks. *Securecomm*, pages 1–5, Aug. 28 2006-Sept. 1 2006.

[16] Jorge Munilla, Andres Ortiz, and Alberto Peinado. Distance bounding protocols with void-challenges for RFID. Printed handout at the Workshop on RFID Security – RFIDSec 06, July 2006.

[17] Maxim Integrated Products. An introduction to direct-sequence spread-spectrum communications. http://www.maxim-ic.com/appnotes.cfm/appnote_number/1890/, 2003.

[18] Kasper Bonne Rasmussen and Srdjan Čapkun. Extended abstract: Implications of radio fingerprinting on the security of sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2007.

[19] Jason Reid, Juan M. Gonzalez Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 204–213. ACM, 2007.

[20] Zafer Sahinoglu, Philip Orlik, and Andreas F. Molisch. Device, dethod and protocol for private UWB ranging. World Intellectual Property Organization, 2007.

[21] N. Sastry, U. Shankar, and D. Wagner. Secure Verification of Location claims. In *Proceedings of the ACM Workshop on Wireless Security (WiSe)*, pages 1–10. ACM Press, September 2003.

[22] S. Sedihpour, S. Čapkun, S. Ganeriwal, and M.Srivastava. Implementation of attacks on ultrasonic ranging systems, demo. *SENSYS*, 2005.

[23] D. Singelee and B. Preneel. Location verification using secure distance bounding protocols. *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 7 pp.–, 7-10 Nov. 2005.

[24] Bernard Sklar. *Digital Communications, Fundamentals and Applications, 2nd Edition*. Prentice Hall PTR., Jan 2001.

[25] IEEE Computer Society. Part 15.3: Wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs). http://www.cmi.ac.in/~sdatta/networks/standards/802.15.3-2003.pdf.

[26] Nils Tippenhauer and Srdjan Čapkun. UWB-based Secure Ranging and Localization. Technical Report 586, ETH Zurich, January 2008.

[27] S. Čapkun, L. Buttyán, and J.-P. Hubaux. SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, Washington, USA, October 2003.

[28] S. Čapkun and J.-P. Hubaux. Securing position in sensor networks. Technical Report 200444, EPFL (Swiss Federal Institute of Technology in Lausanne, 2004.

[29] S. Čapkun and J.-P. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Proceedings of the IEEE Conference on Computer Communications (InfoCom)*, 2005.

[30] S. Čapkun and J.-P. Hubaux. Secure positioning in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(2):221–232, Feb. 2006.

[31] Chin-Der Wann and Hao-Chun Chin. Hybrid toa/rssi wireless location with unconstrained nonlinear optimization for indoor uwb channels. *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 3940–3945, 11-15 March 2007.

[32] R. Ward. Acquisition of pseudonoise signals by sequential estimation. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 13(4):475–483, Dec 1965.

[33] D.P. Young, C.M. Keller, D.W. Bliss, and K.W. Forsythe. Ultra-wideband (uwb) transmitter location using time difference of arrival (tdoa) techniques. *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2:1225–1229 Vol.2, 9-12 Nov. 2003.