Secure Web Services for Low-cost Devices

Johannes Helander
Microsoft Research
jvh@microsoft.com

Yong Xiong
Texas A&M University
yong-xiong@neo.tamu.edu

Abstract

This paper describes how to use XML Web Services and public key cryptography on small devices in consumer settings to achieve a high level of interoperation and security. This is done while maintaining the strict performance requirements that are expected from low-cost devices operating with limited energy and other resources.

1. Introduction

Our everyday lives are filled with various devices that help us with our chores, provide entertainment, help improve our health, provide light and heat, and help us communicate. If we could improve those devices and make them work together—thus enabling new and better devices—our everyday lives would be more comfortable, social, and efficient.

Invisible Computing attempts to do just that. It combines everyday devices with computation and communication capabilities, enabling new functions and aggregation. It does this without forcing us to learn new or archaic computer interfaces. It makes it easy to adopt by keeping costs down and without requiring a pre-existing infrastructure. A watch could be used to control the volume on the radio. The refrigerator could automatically order more beer or notify the TV that its door is ajar. Energy could be saved by integrating sensors with the heating system. Wearable medical devices could improve our well-being. More natural user interfaces and ubiquitous communication would enable better social interaction regardless of our physical location. Smart toys would not only entertain, but also educate children.

In virtually all of these applications we must pay attention to security issues. Installing a home automation system should not result in the loss of privacy. A wearable medical device could benefit from the ability to communicate directly with a doctor's office but should not leak sensitive biometric data to outsiders. Only the authorized physician should be able to tune a pacemaker.

An adequate security model for invisible computing devices must therefore ensure privacy and owner control at all times. To protect privacy we must use strong encryption for any and all communications. To protect owner control we cannot let the compromise of one device compromise the entire system. This requirement mandates a solid key exchange and trust management protocol; one that can only be implemented by a public key infrastructure (PKI), since access to authority servers is not always available.

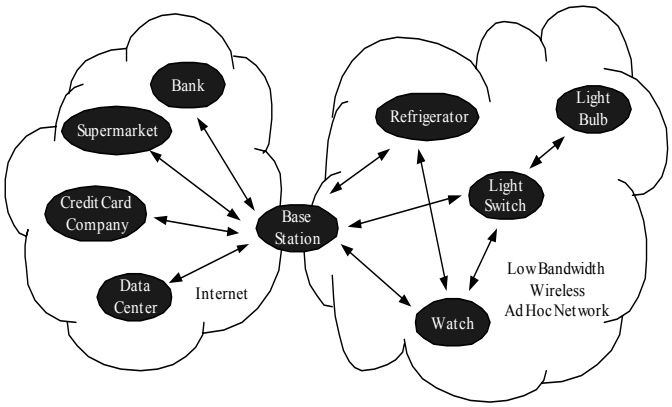

Figure 1. The home as a secure, invisible computing system.

Another issue that invisible computing systems face is interoperability. For instance, a home automation system must integrate cellular phones, refrigerators, TVs, heating systems, watches, heart rate monitors, etc. produced by different manufactures at different times. XML Web Services have been developed to help manufactures overcome this integration barrier. They provide loosely coupled, platform-independent, and language-independent application layer interoperability between different platforms, including small devices.

Combining the security and interoperability requirements with low cost and limited resources is a challenge. Current embedded systems either have no security or provide extremely limited interoperability.

This paper introduces a security model and an implementation that provides a solution to this challenge. It is designed for devices that often must operate in the absence of any infrastructure, communicating in an ad hoc fashion with a small number of peers over unsecured channels, such as wireless radio links or electrical power lines. At times these devices have access to general purpose

computers or the Internet, and want to be part of the global service architecture.

The model provides service discovery, maintenance of trust/function relationships, server/client authentication, secure messaging, and local/global interoperability. It does not assume any global authority or availability of services, but still enables federating with external trust domains. Privacy and owner control are maintained via strong encryption and managed access at all times. Interoperability is achieved through a unified presentation layer that leverages XML Web Services. A base station is required when connecting to the Internet but not for peer-to-peer communication. Setup and discovery are integrated with trust establishment and key distribution—thus creating an easy to use, hassle-free user experience.

We implemented a prototype of this model using standard protocols in an optimized fashion. We used a small number of the well known encryption primitives, RSA, AES, and SHA1; and a combination of the common protocols UDP, XML/SOAP, and Resurrecting Duckling [8]. In the Resurrecting Ducklings protocol, a device adopts the first "mother" it sees as its certificate authority. The implementation runs on a low-cost, single chip micro-controller and performs well enough for real use.

We combine trust establishment and functional assignment so that all setup chores are done in a single user interaction. The interaction is based on physical touch and predicts the correct action heuristically, drawing from any preceding interaction. For instance, first touching a light bulb when it is installed and then a light switch indicates to the light switch that it should control the light and it also gives it the authority to do so.

2. System Architecture

Our secure communication model logically comprises three stages. The first two enable the third.

1. **Trust relation establishment and role assignment:** Who are the participants? This stage deals with questions on how a security domain is established; how an entity is created and what function it should have; how an entity is admitted into a security domain; and how to establish trust across domains.
2. **Discovery and key exchange:** Why and how should we talk? Once an entity wants to talk to another entity it must first find its peer, verify trust, establish a communication key, and decide on communication routes and data representation.
3. **Service communication:** What do we want to say? This stage is where the real work is done. Each entity presents itself as a service that can receive messages. The authenticity of the message needs to be verified and a decision made whether the message should be processed.

Each stage defines a set of protocols and policies. Each stage is explained in further detail in sections 3, Figure 3. , and 4.3. At all stages, basic security properties must be maintained.

3. Trust Management

Trust is established between entities within a trust domain. The trust domain is controlled by an authority that is acknowledged by each member. Each entity is represented by a certificate and the knowledge of a secret that is only known by the entity itself. A certificate consists of the public key (RSA/DSA) that matches the secret and a number of attributes.

A small device—the focus of this paper—has one entity that represents the device. That entity is a member of one trust domain. Conversely, big computers may have multiple entities that represent various services or users on that computer, with each entity potentially being part of multiple trust domains.

In a consumer setting, each person can have her own *personal trust domain*. A personal trust domain consists of all devices that have direct trust relationship even if the device locates in the outside Internet. This means a personal trust domain is not limited to the local ad hoc network. The local ad hoc network can work independently from the outside Internet. Our model also supports occasional secure cross-domain interoperation. Cross-domain trust relations can be established through WS-Federation. Unlike the orange book, our model is not hierarchical, as consumer use and a consumer society do not match a hierarchical model. Our model does not require a global *certificate authority* (CA); therefore systems can be set up completely independently and do not require connectivity.

3.1. Bootstrapping Trust

Trust is established on a different channel than normal communication. In this paper, we call this channel *the secure channel*. This channel could be physical-touch based, meaning that any communication on this channel implies physical presence. It could be a personal visit to the bank, where a teller verifies that the driver's license matches the face. It could be a check with the hash of the public key. When the check clears, trust is established. The reception of money makes the bank trust the customer—the policy of what constitutes trust is up to the participants. The touch based scenario seems most appropriate for consumer electronic devices.

The trust is bootstrapped using the Resurrecting Duckling Protocol on the secure channel. A device becomes a part of the family (the trust domain) of the first "mother duck" it sees. We say the "mother" is the trust authority of all devices in its family. The mother is another device that signs the new device and the siblings believe in the signature because they share the mother. The mother can be viewed as a certificate authority (CA) of the domain. One mother defines a personal trust domain.

This is what happens: Initially a device is alone or "blank". Each device generates a key pair for itself. The mother creates a certificate for itself. A certificate is a public key and a collection of attributes that are signed by the mother's private key (SHA1 and RSA). The blank device sends its public key to the mother. The mother turns it into a certificate and sends it and its own certificate to the device. The device is now part of the family (trust domain). The attributes state what role the device should have, what the identity of the device is, and a contain a static access control list that—together with dynamic policies—provides fine grain control in later stages. Possible attributes include a name, a unique ID, a device category, access privileges, the initial location, associated devices, and the owner of the device.

The device proposes an attribute list, and the mother edits and signs the list together with the public key. The editing process is affected by what the mother touched previously and heuristically reflects the user's intent. For example, a user can touch a light bulb with her mother device first and then physically touch a light switch. The light switch is associated to control the light and is authorized to do so. Not all switches need to be authorized to control all lights even if they all are certified by the same mother. Note that the physical touch is only needed when the light bulb is installed and later configuration can be done remotely if desired.

The mother also creates a shared *house key* and sends it to each of its siblings. This key is later used for discovery question messages to avoid any plain text communication.

3.2. Direct Trust

Within a trust domain, all devices with the same owner trust each other directly. This is subject to the attributes in the certificate since they all trust their mother's signature. Direct trust can take place within the local ad hoc network or cross the Internet.

For the local case, shown in Figure 2. (a), each device has a certificate signed by the same mother and knows the mother's public key after bootstrapping with the Resurrecting Duckling protocol. Thus two devices can authenticate each other by exchanging their certificates.

Figure 2. (b) shows another case. Occasionally, one device wants to request a service outside of the local wireless network through the Internet. For example, a user wants to contact the bank from her watch. To set up a secure connection between the bank and her watch, a trust relationship must be established first. She could take her mother device to the bank and use the touch-based authentication at the bank (the teller checks a picture of the customer and the customer sees the big building and believes it is a legitimate operation). The mother device gives its own certificate to the bank and creates a certificate for the bank by signing the bank's public key and some attributes that state that this is the bank for the given account. The Resurrecting Duckling Protocol is used to achieve this. The process is essentially the same as when the mother admits a light switch.

The public keys could also be exchanged in other ways. In fact, rather than sending the entire key, a hash value is sufficient. The customer could send a check with a hash of the public key. The bank could publish a hash of their key in a major newspaper. The attributes can be established manually based on the context. In a low-tech bank branch, the trust could be exchanged with ID cards and pieces of papers with the key hashes on them.

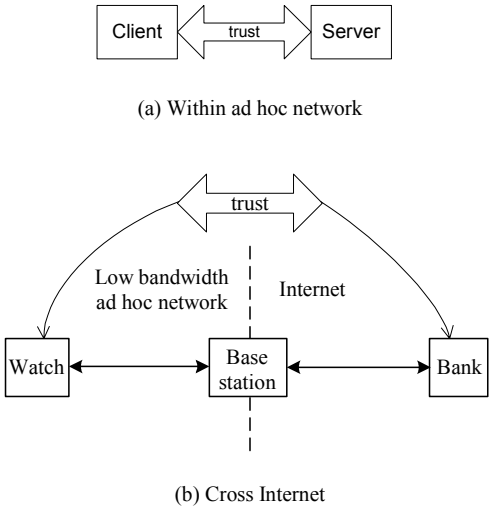

Figure 2. Direct trust between devices within a personal trust domain.

The bank can later use the hash of the mother device's public key to verify that a certificate sent to it is correct and signed by the mother. For example, a watch trying to contact the bank through a base station and the Internet will present its own certificate, send a copy of the mother's certificate, then request the bank's certificate (or use the manually entered certificate that came on a piece of paper).

3.3. Indirect Trust

In some cases, a device needs to request a service across trust domains. For example, the refrigerator wants to order some beer from a supermarket. The supermarket is not within the same trust domain as the refrigerator. There is no direct trust relationship between the refrigerator and the supermarket, but the refrigerator and the credit card company trust each other. The supermarket and the credit card company also trust each other. When the refrigerator wants to access the service of the supermarket, they talk to each other first to find which identity is trusted by both of them (step 1 in Figure 3.). Then the refrigerator gets the public key of the supermarket from the credit card company (step 2 in Figure 3.), as does the supermarket (step 3 in

Figure 3.), thus creating trust between the refrigerator and the supermarket.

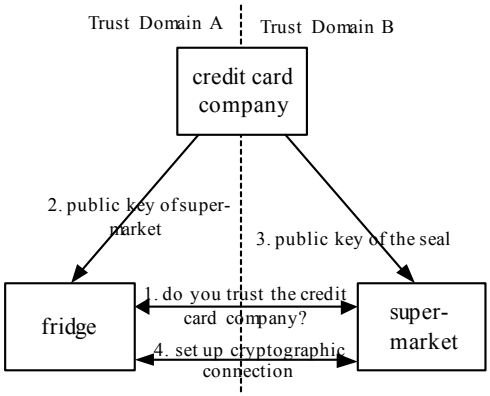

Figure 3. Cross domain trust.

The above case matches the Federated Identity Model of the industry-standard *global XML architecture* (GXA). In GXA terms, public keys are *security tokens*. A device acts as its own *security token service* instead of using a separate service. The level of trust established indirectly is less or equal to either direct link.

4. Discovery and Key Exchange

The purpose of this stage is to establish enough state to allow communication between two parties. The instigator of communication needs to find the proper entity to communicate with, authenticate the entity, establish a communication key, find a communication path to the entity, and negotiate the protocol and data representation. All communication in this stage is done on the public channel and is categorically encrypted.

For each peer an entity wants to communicate with, it caches a certificate with RSA keys and attributes, a URL, a peer-to-peer AES and SHA1 key, a network route to where the peer was last seen, a sequence number, a session ID each way such as a UDP port number, and parameters (WSDL, compression, preferred protocol). Any of the state can be arbitrarily discarded with the cost of having to redo part or all of the discovery.

4.1. Service Discovery

If the client wants to access a service, it will check if the URL of the service is cached. If not, it will send a *discovery request* through an IP multicast SOAP message on the public channel. The discovery request message gives a description about the requested service. This might be a DNS host name or something more abstract. The server sends back to the client a *discovery response* containing its URL through a unicast SOAP message. Potential service masquerading is detected by checking the certificate and ACL of the replying entity.

Sometimes a user needs to access a device from outside of her personal ad hoc network (e.g. the home). In this case, the user sends a service request to the base station of her personal network through unicast, and then the base station multicasts the service request within the wireless ad hoc network. The right service will send a *discovery response* to the base station and the base station will forward it to the client as a regular internet directory service. The message forwarding is done with Web Service routing and the SOAP protocol. If the base station has a cache of the record of the required service, it could respond to the client directly. In this scenario, the base station acts as a discovery service.

Conversely, to find services that are outside the local ad hoc network from within the local network, the base station again acts as a proxy. The base station can either reply to the discovery message on behalf of the service, or the device can start by discovering a directory service and then use that for further queries.

All discovery messages (SOAP messages) are encrypted with a home key. The home key is a symmetric (AES) key generated by the mother device and is common to the security domain. Its purpose is to keep curious neighbors from knowing what is being discovered.

4.2. Two-Way Authentication and Key Exchange

After a potential peer has been located, it is time for authentication and key exchange. Both parties authenticate each other. We use a combination of symmetric and asymmetric key encryption techniques that keep the asymmetric use at a minimum, since it is much costlier to calculate. The two-way authentication and peer-to-peer key exchange is done with RSA, while the connections are encrypted with an AES peer-to-peer key.

Authentication involves exchanging and verifying certificates. In a certificate, the public key of a device is the most important. It also contains the attributes of a device. Basically, authentication with PKI is a process to verify if a public key belongs to the right entity. If a public key is verified to belong to an entity, then one can trust that any information encrypted with the public key can only be understood by the entity with the right private key. The attributes are used to check against an *Access Control List* (ACL) to do authorization.

After the server and client authenticate each other, the client generates a random symmetric peer-to-peer key, encrypts it with its own private key and the server's public key (contained in the server's certificate), and sends it to the server. The server decrypts it with the client's public key (contained in the client's certificate) and its own private key. Caching the results turn these public/private key operations into one-time events.

4.3. Service Communication

Once a shared peer-to-peer key is created, an encrypted connection can be set up between the client and the server. Encryption is applied to SOAP messages instead of transport layer packets like SSL does. Every node is a server and every node accepts SOAP messages.

There are four basic cases of communication patterns:

1. Direct communication between peers in an ad hoc network.
2. Communication to the outside where the base station is trusted with the data.
3. Communication with the outside where the base station is not trusted.
4. Communication within the ad hoc network through an intermediary.

In all cases, communication comprises encrypted SOAP messages. How the messages are encoded and what parts of the messages are encrypted with what keys vary slightly.

4.4. Peer-to-Peer Messages

The discovery process established a peer-to-peer key and a session identifier (SID). The sender calculates a digital signature (HMAC) and attaches it to the message. The sender then encrypts the entire message using symmetric encryption (AES) and attaches the SID and sequence number in plain text to the beginning of the message. The encryption is driven by a sequence number and the peer-to-peer key.

The receiver gets the message and uses the SID to look up the correct peer state (if none is found the message is discarded). The received sequence number is compared with the expected sequence number and if it is smaller, the message is assumed to be a duplicate and is discarded. The expected sequence number is, however, only updated once the message has been accepted so as to prevent simple denial of service attacks.

Next, the receiver applies the sequence number and peer key to decrypt the message. It then calculates a HMAC checksum using the peer key and compares it to the one received. If the checksum does not match, the message is discarded.

Finally the receiver checks the message against the attributes of the peer and decides whether the message should be processed in light of access control lists and possible recall lists, etc. If the message can pass through all the filters, it is processed by the SOAP deserializer and served by the correct server object.

The reply is sent back to the client with information in the SOAP routing header that lets the client correlate the response with the request. All the encryption and verification is done exactly like the request, but sent in the opposite direction.

Note that it is safe to transmit the SID and sequence number in plaintext. Any forgery inevitably leads to the message being discarded by the receiver as the AES and HMAC calculations will not yield the expected result. The information content of these numbers is low enough that it does not constitute a privacy threat. The SID could be a UDP or TCP port number thus saving a couple of bytes.

4.5. Messages through Trusted Base Station

Communication with the outside world entails interoperability and a base station. The simplest case is where the base station is trusted enough to handle low security operations and can offload complications from a device. An example scenario is a refrigerator that wants to order milk from the supermarket. The refrigerator is no more secure than the base station so the base station can order the milk on behalf of the refrigerator.

The refrigerator authenticates itself to the base station and uses the peer-to-peer communication patterns. The base station then uses the WS-Security and WS-Federation Internet Web Service protocols encoded as XML and Base64 blocks to order milk from the supermarket, while the wireless side simply encrypts the entire message. The two communication patterns and trust relationships are created independently. This scenario precisely corresponds to the *Passive Profile* of WS-Federation.

4.6. Messages through Untrusted Intermediary

In some cases, the security of the base station is insufficient; therefore an end-to-end secure channel must be created. The base station is still needed for forwarding messages. This corresponds to the *Active Profile* of WS-Federation.

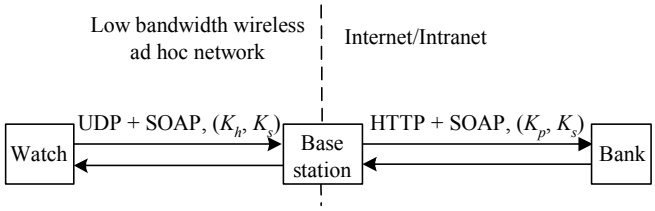

Figure 4. A watch sends messages to a bank.

The scenario in Figure 4. is an example of this. The watch sends a SOAP message to the bank through a base station. The watch is within the wireless network and the bank is on the outside Internet. The base station knows how to forward the message to the bank. The base station needs to access the SOAP headers but only the bank and the watch need access to the body.

The SOAP body is encrypted with the end-to-end (watch to bank) peer key (K_s in Figure 4.). Only the endpoints can understand the contents of the body. On the wireless link the SOAP header is encrypted with the peer key that is shared between the base station and the watch (K_h in Figure 4.); while between the base station and the bank, the SOAP header is encrypted with a peer key (K_p in Figure 4.) that is shared between the base station and the bank.

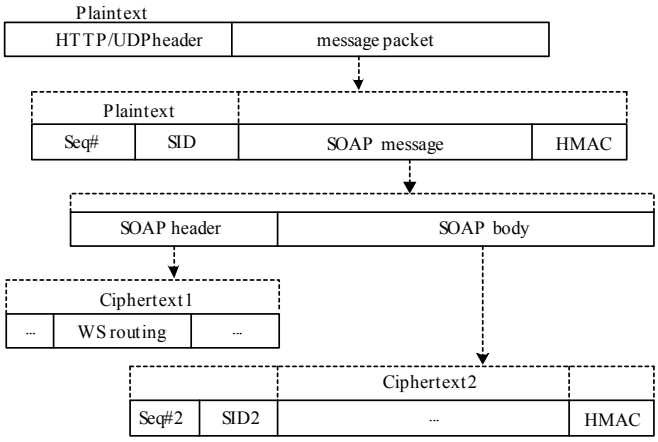

Figure 5. A routed message packet

SOAP messages can theoretically be sent over any transport. Our implementation supports TCP/HTTP and UDP. Within the wireless network, SOAP messages are transmitted through UDP instead of HTTP. UDP is cheaper than HTTP and the extra headers are not needed. The base station then forwards the SOAP message to the bank through HTTP. The message forwarding is done using a Web Service routing SOAP header.

Figure 5. shows the format of a message packet. Since it pointless to encrypt the same data twice, the "inner" packet is only encrypted with the end-to-end key; while the "outer" packet is encrypted with the point-to-point key. The outer HMAC, however, covers the entire message to maintain integrity between the halves.

4.7. Ad Hoc through Intermediary

This case is essentially the same as communication with the outside. It may involve the base station as a more powerful and possibly trusted intermediary or it may involve any device that happens to be able to do forwarding. Security can be constructed either end-to-end or hop-to-hop as needed.

5. Impementation and Performance

We implemented a prototype of the model described in this paper on a research RTOS from Microsoft Research. The RTOS is specifically designed for invisible and embedded use, is built out of object oriented components, and takes a pay-as-you-go approach to composition.

The implementation uses SOAP as a general communication protocol. For example, we use SOAP to do key distribution, service discovery, two-way authentication, etc. The implementation interoperates with ASP+ and the SOAP Toolkit on Windows XP®. The SOAP implementation supports high level remote method calling. Through automatically generated SOAP proxies it is almost as easy to call a remote method as it is making a function call in a local application, with some obvious differences in timing and failure modes.

We measured our system on AT91EB63 evaluation boards (called EB63 board for simplicity in the rest of this paper). An EB63 has a 25 MHz ARM7 microcontroller, 256 Kbytes SRAM and 2 Mbytes Flash. Since this is more than a cost-effective system would have, we limited the memory usage to 32 Kbytes of RAM and 256 Kbytes of Flash ROM. The board was chosen for its processor and I/O capabilities, not for its energy efficiency or lack thereof.

Instead of an actual wireless network we ran the measurements over serial lines with one serial line representing a secure channel and another representing the public channel. The serial lines were run at 38400 baud, which is in range of several available low-power wireless radios. We used a PC as a "base station" achieving connectivity to the Internet.

We will now evaluate the cost of the different pieces based on measurements. The cost includes costly system resources, time, and energy.

TABLE I. FOOTPRINT (ARM - IN BYTES) AT PEAK USAGE

Files	ROM	Static RAM	Heap	Stack	Total RAM
BASE	24,676	1,940	2,837		2,777
DRIVERS	11,464	332	896	2,288	3,516
NET	77,024	3,424	2,648	3,400	9,472
XML	7,860	16	88		104
SOAP	29,504	280	996	4,320	5,596
SECProto	14,180	604	1,848	2,648	5,100
AES	16,532	8			8
RSA	9,784	28	24		52
SHA1	5,436	8			8
C-Library	7,620	12			12
TOTAL	**204,080**	6,652	9,337	12,656	**28,645**

5.1. Footprint

The system can be compiled with many compilers. The measurements were carried out using the ARM Software Development Kit 2.11. TABLE I. shows the memory usage of the whole system. The ROM footprint is the amount of Flash required. The RAM footprint is measured at the point of execution where the memory usage was at its maximum. The RAM usage of the individual components varies but this is the point that determines how much actual RAM is required.

Figure 6. and Figure 7. show the percentage occupation of footprint. The network stack occupies about 38% of the footprint of the whole system. It includes DHCP, IGMP, IP, UDP, multicast, routing, sockets, etc. The code also supports IP auto-configuration for IP applications on an ad hoc network in the absence of a DHCP server.

The XML parser and generator take about 8 Kbytes or 4% of the total. The SOAP component includes a schema checker and a serializer/deserializer that translates between SOAP messages to application stack frames according to an XML specification.

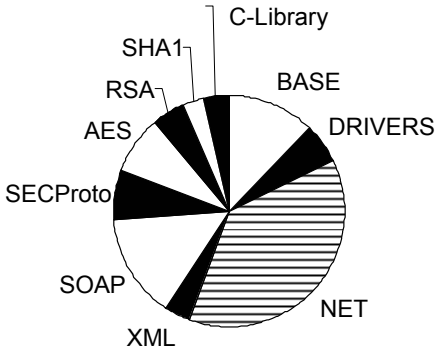

Figure 6. ROM Footprint.

The SECProto component leverages the SOAP component and includes the security protocol implementation, i.e. trust management, service discovery, key distribution, two-way authentication, etc. The cryptographic algorithms AES, RSA and SHA-1 are lifted from Windows® and are not particularly optimized for size. The C-Library is the part of the ISO C runtime library that was used.

The BASE component includes the real-time scheduler, a heap manager, a loader, any machine dependent initialization code, threading and synchronization, and the unified namespace. The total system heap is 9 Kbytes including the usage of the heap itself but excluding stacks. We can see that the ROM footprint (including code and read only data) of the whole system is less than 200 Kbytes and at peak usage the RAM footprint—including static data (.data, .bss, and interrupt vectors), heap and stacks—is 28 Kbytes.

5.2. Latency

We measured how long it took to respond to a service message. TABLE II. shows the latency of major cryptography operations. We ran each operation on an EB63 board—running from slow external Flash memory—500 times and calculated the average latency and the standard deviation. Generating a key pair of 1024-bit RSA takes almost 5 minutes. Fortunately, key pair generation is required only once on each device, when first initialized.

Encryption/decryption with an RSA private key is also quite expensive. It takes about 103 seconds per kilobyte. There are two applications of RSA: 1) signing the hash value of a certificate and 2) exchanging a peer-to-peer key. For both of these cases, the data is less than a block (128 bytes). A certificate only needs to be signed once. Any later use involves the cheaper public key operation. The latency of signing a certificate is thus about 14 seconds.

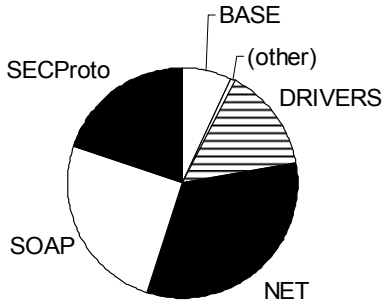

Figure 7. RAM footprint at peak usage.

Exchange of one peer-to-peer key needs four RSA cryptographic operations: Two with private keys and two with public keys. The first private key operation can be pre-calculated so that it does not need to be factored into the response time. Therefore, the cryptographic latency of peer-to-peer key exchange is about 14 seconds. If one of the parties is a more powerful computer, it can bear the burden of both expensive operations by a combination of RSA and DSA use. That cuts the exchange time down to about 1.5 seconds.

The most frequently used cryptographic operations are AES encryption/decryption and SHA1-HMAC. TABLE II. shows that encrypting/decrypting one Kbytes with 128-bit AES takes about 16.3 ms and hashing one Kbytes with SHA1-HMAC takes about 79.6 ms. However, in AES CTR mode, counters can be predicted easily so that encrypting the counters with an AES key can be pre-calculated during CPU idle time. Thus the latency of CTR-mode AES cipher is just XOR operation time. XOR operation latency is negligible. Therefore, the latency to encrypt a message with one Kbytes and hash it is just 79.6 ms.

TABLE III. shows that the latency of doing one remote ADD operation through SOAP. The SOAP request message for ADD operation is 835 bytes long and the SOAP response message has 747 bytes. Including ~74 bytes of overhead (14-byte Ethernet header, 20-byte IP header, 8-byte UDP header, and 20 bytes for HMAC, 4 for Sequence number and 16-byte alignment) for each of them, the serial packet for the SOAP request message is 912 bytes, and the SOAP response message is 818 bytes. Transmitting one byte on serial line needs 10 bits. Therefore, the theoretical serial transmit latency = (912 + 818) * 10 / 38400 / 1000 = 450 ms, where 38400 is the baud rate of the serial line.

5.3. Energy Consumption

Energy consumption is directly related to: a) how much data has to be transmitted—the time the radio is on, and b) the amount of computation that needs to be done—the time

the CPU is on. Some of the cost could be alleviated by compressing the messages so less is transmitted.

The overhead of the secure protocol is, on average, 30 bytes per service message compared to plain text messages. This is about 4% of the XML messages. With a compressed message, the overhead would be somewhat higher.

The CPU overhead of the encryption per message with the same dataset as the latency calculation is about 20%, excluding the one time costs of certificate handling and key exchange. This is the percentage of the message processing—the percentage of the total workload of the system depends on applications that run on the platform. We measured the EB63 board energy usage and observed that it consumes 68 mA at 7V when idle and 108 mA when busy. Integrating the difference for one service request yields 270 mJ, which corresponds to 20 million cycles. More energy efficient hardware would yield smaller numbers. The measurement, however, reflects the finding that the protocol processing can be done within reasonable time and within a reasonable number of cycles.

TABLE II. LATENCY OF CRYPTOGRAPHY OPERATIONS

Algorithm	Operation		Latency on a 25 MHz ARM 7		
			Average	Standard deviation	Per Kbyte
1024-bit RSA	Generate a key pair		290 s	56%	N/A
	Private key	Encrypt/decrypt a block (128 bytes)	12.9 s	<1%	103 s
	Public key	Encrypt/decrypt a block (128 bytes)	0.667 s	<1%	5.34 s
128-bit AES	Encrypt/decrypt a block (16 bytes)		0.254 ms	<1%	16.3 ms
SHA1-HMAC	1024 bytes		79.6 ms	<1%	79.6 ms

TABLE III. LATENCY OF REMOTE ADD OPERATION

	Latency on a 25 MHz ARM 7		Processes included
	Average	Standard deviation	
Total measured latency	760 ms	5%	Generate, parse, process, encrypt/decrypt and transmit the SOAP request and response.
Theoretical serial transmit latency	450 ms	N/A	Ideally transmit the request and response on serial line.
Local SOAP processing latency w/o encryption	101 ms	2.1%	Generate, parse and process the SOAP request and response.
Cryptographic latency	65.6 ms	<1%	AES and SHA1-HMAC on the request and response.
Other	143 ms		Drivers, network stack, etc.

6. Related Work

Many researchers identify that trust management plays a significant role in a distributed security system [1], [2], [3], [4], [5]. Yahalom *et al.* gave a formal definition of trust [6]. Wilhelm *et al.* discussed trust management for mobile networks in [7]. A trust bootstrapping protocol, the Resurrecting Duckling protocol, is proposed in [8]. It avoids an online CA and is suitable for low-cost device use. We extend the Resurrecting Duckling protocol to functional relationship initialization and trust federation.

Tatebayashi *et al.* proposed a key distribution protocol (TMN) for mobile network [9]. However, their protocol is only suitable for star-type mobile networks and some researchers point out it is flawed, e.g. Simmons describes an attack against TMN [10] and Park *et al.* also analyzed its weakness and propose an improvement for it [11]. Carman *et al.* compared performance of a wide variety of key distribution schemes on different sensor network platforms [12].

Beller and Yacobi propose a protocol that uses pre-computation techniques to reduce the response time of key distribution for mobile uses [13]. However, their protocol is vulnerable to a man-in-the-middle attack [14].

Zhou and Haas use routing redundancies of ad hoc networks to achieve availability, and use threshold cryptography to isolate compromised nodes [15]. Marti *et al.* proposed a mechanism that uses a watchdog to recognize misbehaving nodes and then uses a *patherater* to avoid them.

Hubaux, Buttyan and Capkun analyze security threats specific to ad hoc networks and propose a self-organizing public-key distribution scheme, in which certificates are issued by users (corresponding to devices in our work) instead of a CA [16]. Their algorithm involves complex graphic operations that are neither scalable nor suitable for embedded systems use.

Czerwinski *et al.* propose a secure centralized service location model, in which service advertisements and queries are all done through a central server [17]. However, their work is too complicated for use on low-cost devices.

All the above papers involve one or several aspects of the security issue for low-cost embedded system use. The

model proposed in this paper pulls them together and fills in the gaps.

Fox and Gribble presented a security protocol for mobile computing based on a proxied version of Kerberos IV, which provides secure access to application level services [18]. However, as would be expected, their solution requires an online centralized authentication service. Traditional security solutions that require online trusted authorities or certificate authorities are not suited for mobile ad hoc network environments. Mobile ad hoc networks are often unable to provide access to an online centralized trust authority due to their highly dynamic infrastructure and their need for reliable autonomous operation.

Perrig *et al*. [19] propose two secure building blocks for low-cost devices: SNEP and μTESLA. They claim that SNEP provides confidentiality, authentication, and data freshness, and μTESLA provides authenticated streaming broadcast. Their system security is based on a preset master key shared by all devices. The block chaining used does not remove the weakness, that if an adversary compromises the master key on any device, he can easily eavesdrop and impersonate all other devices. In our system, we use PKI to exchange trust and peer-to-peer keys and avoid these pitfalls, but end up with a slightly more complex model. The savings from avoiding the trust and key issues might not be significant enough for the smallest embedded systems as there still is no space left for applications in [19].

We introduced embedded Web Services in [20]. No other comparable work has been published since. In this paper, we focus on secure Web Services for invisible computing use. As far as we know, even though recently much work has been done on ad hoc network security, nobody else has done any research on secure Web Services for embedded systems.

Our work shows two suppositions are incorrect: 1) Web Services would be unsuitable for embedded use because they need a large footprint, CPU time, energy, and network bandwidth; and 2) that public key infrastructure (PKI) would be unsuitable for embedded use because it consumes lots of energy and CPU time [19].

One may argue that Web Services' advantages come with a performance penalty: XML based SOAP messages are textual so that their sizes are significantly larger than protocols that send specific binary data. It turns out that the special protocols are often not that efficient and their inability to scale to new demands make it necessary to support many different mechanisms largely erasing any performance benefit. We also note that compressing XML can be done in CPU-efficient ways and result in significant reduction in size.

7. Conclusions

This paper described a secure communication model and implementation for invisible computing. It showed that it is possible to combine low-cost with strong security and first class interoperation. A trust and key exchange model based on public key infrastructure and a presentation layer based on XML Web Services were not out of reach when properly put together.

Combining trust establishment with functional assignment led to a physical touch based user interaction paradigm that did not completely eliminate configuration, but made it simple and understandable. Federating with outside trust authorities proved centralized and hierarchical models unnecessary. The independence achieved allows for incremental and self-sufficient deployment.

The security does not come for free but the cost is in our view reasonable considering the alternative of inadequate security. A high level of security and interoperation is achievable on low-cost devices and should therefore be adopted.

The implementation proved that secure Web Services make sense in embedded systems with a careful design and on-the-target optimizations. The disciplined component-based approach led to efficiency and rapid development.

Reference

[1] DoD Trusted Computer System Evaluation Criteria, 26 December 1985 (Supersedes CSC-STD-001-83, dtd 15 Aug 83). (Orange Book)

[2] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *Proceedings of the IEEE Conference on Privacy and Security*, 1996.

[3] P. Zimmermann, "PGP User's Guide," *MIT Press*, Cambridge, 1994.

[4] U. G. Wilhelm, S. Staamann, and L. Buttyan, "On the problem of trust in mobile agent systems," in *IEEE Network and Distributed Systems Security Symposium 1998*, pages 11-13, San Diego, CA.

[5] R. Yahalom, B. Klein, and T. Beth, "Trust relationships in secure systems—A distributed authentication perspective," in *Proc. of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 150-164, May 1993.

[6] R. Yahalom, B. Klein, and T. Beth, "Trust relationships in secure systems—A distributed authentication perspective," in *Proc. of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 150-164, May 1993.

[7] U. G. Wilhelm, S. Staamann, and L. Buttyan, "On the problem of trust in mobile agent systems," in *IEEE Network and Distributed Systems Security Symposium 1998*, pages 11-13, San Diego, CA.

[8] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," *LNCS 1796*, Springer-Verlag, 1999.

[9] M. Tatebayashi, N. Matsuzaki, and D. B. J. Newman, "Key distribution protocol for digital mobile communication systems," in *Advances in Cryptology-Crypto '89 Proceedings*, Lecture Notes in Computer Science, vol. 435, 1989, pp. 324-334.

[10] G. J. Simmons, "Cryptanalysis and protocol failure," *Communications of the ACM*, 37(11), Nov 1994.

[11] C. Park, K. Kurosawa, T. Okamoto, and S. Tsujii, "On key distribution and authentication in mobile radio networks," in *Advances in Cryptology EuroCrypt '93*, Lecture Notes in Computer Science, vol. 765, pp. 461-465, 1993.

[12] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," NAI Labs Technical Report #00-010.

[13] M. J. Beller and Y. Yacobi, "Fully-Fledged Two-Way Public Key Authentication and Key Agreement for Low-Cost Terminals," *Proceedings of Electronic Letters*, May 27, 1993, Vol. 29, No. 11, pp. 999-1000.

[14] C. Boyd and A. Mathuria, "Key establishment protocols for secure mobile communications: A selective survey," in Proceedings of ACISP'98 , Lecture Notes in Computer Science, vol. 1438, 1998, pp. 344-355.

[15] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Network Magazine*, 13(6), 1999.

[16] J. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing* (MobiHOC) 2001.

[17] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An Architecture for a Secure Service Discovery Service," in *the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (*MOBICOM 1999*), pages 24-35, Seattle, WA USA, August 1999.

[18] A. Fox and S. D. Gribble, "Security on the move: indirect authentication using Kerberos," in *the 2nd Annual ACM/IEEE International Conference on Mobile Computing and Networking* (*MOBICOM 1996*), pages 155-164, White Plains, NY USA, November 1996.

[19] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Wireless Networks Journal* (WINE), September 2002.

[20] A. Forin, J. Helander, P. Pham, and J. Rajendiran, "Component Based Invisible Computing," in *the 3rd IEEE/IEE Real-Time Embedded Systems Workshop*, London, December 2001.

[21] R. Troll, "Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network," http://www.ietf.org/internet-drafts/draft-ietf-dhc-ipv4-autoconfig-04.txt.

[22] A. Abdul-Rahman and S. Hailes, "A distributed trust model," in *Proc. New Security Paradigms Workshop* (*NSPW-97*), New York: ACM, 1997, pp. 48-60.