A new Secure Virtual Connector approach for communication within large distributed systems

Gabriel Risterucci Traian Muntean Léon Mugwaneza
ERISCS Research Group*
I2M-UMR7373 CNRS, LabEx ARCHIMEDE
Aix-Marseille Université
(firstname.lastname@univ-amu.fr)

***Abstract*—Communicating entities in distributed systems and large scale applications require specific message exchange protocols which can be adjusted for multiple networks. Some secure networking protocols exist and provide different security properties. Such protocols include Transport Layer Security (TLS) and Secure Shell (SSH).**

We propose here a more specific approach for constructing a new model of distribution using connectors which implement a protocol as a support for securing exchanges over heterogeneous networks used for distributed applications. The Secure Virtual Connector (SVC) protocol provides enhanced security for exchanges between components of distributed applications. This protocol avoids existing shortcomings within existing secure communications protocols which have been designed to fit a wide variety of situations. This flexibility leads to potential vulnerabilities; most of which are avoidable. We consider here a full set of essential security properties for large distributed application such as confidentiality, authenticity, and a certain form of privacy. Other considerations include the use of heterogeneous networks, as well as the mobility of users using secure virtual connectors. The SVC protocol proposed here provides all the required security properties while keeping a low performance overhead which makes it efficient for both fixed and mobile networks. As such SVC is a suitable alternative to existing secure communication protocols.

I. Introduction

Complex exchanges between objects (processes, components...) is the main aspect of concurrency when dealing with distributed or parallel computing [17], [18]. One can specify basic communications from several points of view. For instance primitives interactions can be synchronous or asynchronous, point-to-point or some form of broadcasting message exchange protocols. The theory behind point-to-point communication is well-established in process algebra [4]. On the other hand more complex and higher level communication schemes encountered in many applications and programming models remain poorly represented in the theory of distributed computing. We emphasize [7], [8] that group interactions within heterogeneous systems shall be considered as a more appropriate exchange scheme for modeling and reasoning about modern communicating systems and networking applications (e.g. mobile computing, data and knowledge mining, cloud computing, etc). Together with others we strongly believe that a theory of communicating systems shall be further developed on the basis of constructed correct interactions protocols between components which we call *virtual connectors*, which encapsulate various exchange protocols including diffusion based communications between parallel or distributed components. In this paper we deal only with basic connectors for point-to-point based exchanges and we propose a security model for such Secure Virtual Connector (SVC).

The main security properties required for secure communications protocols are confidentiality, integrity, and authenticity. Confidentiality means that the contents of messages are illegible without authorization. Integrity and authenticity ensure that no unexpected modifications took place during an exchange. Authenticity is also used to reliably identify the sender of a message. Secure communications relies mainly on two protocols: Transport Layer Security (TLS) and Secure Shell (SSH). While these protocols provide the required properties for secure communications they have shortcomings and issues [23]. Examples of such problems are the use of unsafe encryption schemes [24], dependencies on unprotected user settings [12], vulnerable software, and implementations flaws [10].

TLS and SSH's issues exist because these protocols can operate in multiple scenarios. When different, unrelated software modules have to communicate with each other it is important to have interoperability as well as backward compatibility. This is why protocols like TLS are useful. Server and client software are able to communicate even if one of them is outdated or operates on a constrained system. In addition, neither side of the communication can make assumptions about what protocols and algorithms the other side supports. However, there are situations where we can safely make some assumptions. For example, with communication between instances of the same application one can assume that there will be no version mismatch or unexpected requests. It makes sense then to focus on essential security properties and avoid features that are not required.

We have examined real world requirements for communication between instances of distributed application spanning across heterogeneous networks, and designed a protocol for secure communications compliant with these requirements.

**This work is partially supported by SESIN SA (www.sesin.com) for specific implementations developed for e-health systems and applications (G. Risterucci's forthcoming PhD)*

This new protocol, called Secure Virtual Connector (SVC) also avoids non essential features to prevent known threats from affecting it. To this end, we designed a protocol that provides confidentiality, authentication, and integrity, as well as concealment of the identity of the users. This extra protection allows a client application to securely communicate with a server while never disclosing the identity of the user. While being fully secure, the protocol keeps a low network overhead, both in the size of extra data needed to communicate securely, and in the number of round trips needed to establish a secure connection. This last point is important for mobile communications where connection loss and further renegotiation are a concern.

Since we designed the SVC protocol for specific usages, it is less prone to weaknesses linked to the flexibility of other protocols, and provides an additional security feature in the form of the protection of the privacy of the user.

The remainder of this paper is organized as follows: Section II describes TLS and SSH which are the two main secure communication protocols used nowadays. We give an overview of their initial purpose, the security they provide, and their shortcomings. Section III summarize our main motivations for this work, and provide an analysis of the constraints we considered when designing the SVC protocol. Section IV outlines the SVC protocol, including how it operates, the security properties it provides, and how these properties hold against common attacks. Section V concludes on an evaluation of the efficiency of SVC regarding two common network types: Bluetooth and TCP/IP.

II. Existing secure communications protocols

The two major protocols used for securing communication at application level are Transport Layer Security (TLS) and Secure Shell (SSH). They both provide at least confidentiality and authenticity of the server. We briefly present TLS and SSH, and some shortcomings in our use case, namely communication between instances of the same application.

From this point onward, the terms *client* and *server* are not referring to the general model “one server, many clients”, but simply refer, respectively, to which entity initiated a network connection, and which one was ready to accept the corresponding incoming connection. We therefore also consider peer-to-peer exchanges when using these terms.

A. TLS

TLS is the successor of Secure Sockets Layer (SSL), initially introduced by Netscape [9]. Although similar there are differences that make TLS and SSL incompatible. The two primary goals of TLS are secure communication and interoperability, as stated in the corresponding RFC [6]. TLS uses cryptography to provide secure communication. This includes encryption algorithms, secure key exchange, and digital signature. The interoperability comes from supporting multiple algorithms and having fall-back conditions to support previous versions (including previous SSL versions). These older versions are still extensively used by both clients and servers applications.

TLS provides confidentiality of an exchange between a server and a client, as well as server authenticity. Client can authenticate the server without prior knowledge about the server. Servers can request client authentication based on X.509 certificates [5]. When the client sends his own certificate to the server, this is done without encryption.

TLS acts as an intermediate layer between an application and the actual protocol used by the application to exchange data. For example, TLS is used in the Hypertext Transfer Protocol Secure (HTTPS) protocol [22] to transport regular Hypertext Transfer Protocol (HTTP) requests. TLS transparently takes care of the security of the communication. In practice, TLS operates with protocols that do not take care of the security of their communications. Another major use of TLS is with Virtual Private Network (VPN) software where it transports actual VPN packets to the server as in HTTPS.

The last version of the TLS protocol (version 1.2) can provide a high level of security by restricting the available algorithms to *safe* ones and using secure key exchange. However, issues persist regarding TLS. Some issues exist with the protocol itself and can not be mitigated without removing retro-compatibility with the affected versions of the protocol. Other issues depend on the software environment in which TLS operates [15].

One of the main issues regarding TLS is that it is commonly used with a Public-Key Infrastructure (PKI) that depends on local configurations. More specifically, the authentication of the server usually depends on the validity of a certificate, checked with a list of *known* authorities. Since numerous applications take these known authorities from a store managed by the Operating System (OS) on the client side, an attacker can infect this store and remove the element of trust provided by certifications authorities.

In some environments there are implementations of TLS readily available and provided by the OS. This allows easier application deployment but also ties the security of the application to the OS security features. Receiving OS updates is not always feasible especially on embedded and mobile devices. When new threats appears, applications that depend on obsolete software for their security become vulnerable.

In addition to issues related to the usage of TLS and its implementations there are issues with the TLS protocol itself. Attacks have been reported on all versions of TLS [23]. Most pressing issues of TLS are the lack of anonymous client authentication and backward compatibility with older versions of both TLS and SSL. Backward compatibility causes some previous attacks to remain effective despite corrections in later versions of the protocol [19].

B. SSH

The SSH protocol is widely used as a way to interact with a distant system by opening a shell over a network connection, but it has other uses. In fact, SSH is a combination of three parts that allow it to operate as a secure transport layer for different kind of applications [26].

The first part is the transport layer [28], which provides secure bidirectional communication between the client and the server, as well as server authentication. Server authentication is host based, and can use a variety of methods, including public-key authentication. The second part is the user authentication protocol [25]. The authentication operates over the SSH transport layer and the user's credentials remains confidential. This protocol provides multiple methods of authentication, including public-key signatures. The final part is the connection protocol [27]. This component allows actual communication at the application level. The connection protocol can handle multiple channels used by the application to communicate different types of information. For instance, there can be a channel for using a shell, and a second channel to perform IP forwarding between the client and the server. All these channels are then sent through the SSH transport layer which is in charge of ciphering and integrity checking.

Few known security issues exist regarding the SSH protocols itself. Most of them date from the first version of the protocol [1], and are obsolete since the second version of the protocol (SSH-2). Remaining issues are, as in TLS, caused by the possible use of weak cipher algorithms [3]. The mitigation technique is the same and consists in removing weak algorithms from the negotiation step. Another issue is the complexity of integrating SSH support in some applications. The most popular implementation (OpenSSH) must run as a separate process which is sometimes not possible. Other implementations exist but they may pose portability or licensing issues.

III. Motivations for the design of a new secure exchange protocol

A. Security properties

From cloud-enabled software to large distributed applications, modern software heavily depends on network exchange protocols. These exchanges can contain private or sensitive information and thus require appropriate security mechanisms. Moreover, such exchanges happens through heterogeneous networks like fixed networks and mobile wireless networks.

We consider here two types of communication, which have different constraints: external communications between independent applications (for example a web browser and a HTTP server), and internal communications between components within multiple instances of the same application.

The external communication model often requires interoperability between software from different sources even sometimes using different versions of the same protocol. In this model there is a strong emphasis on backward compatibility and availability of the algorithms used by the communication protocol. The developer of an application can only make weak assumptions regarding the kind of software that his application will interact with. In protocols fitting this generic model there is a trade off in favor of availability that is detrimental to the security.

TABLE I
Properties of different protocols

'–': missing property, 'O': optional property, '+': mandatory property

Properties	TLS	SSH	SVC
Confidentiality	+	+	+
Forward secrecy	O	+	+
Server authentication	+	+	+
Client authentication	O	+	+
Identity protection	–	+	+
Reduced attack vectors	–	–	+

The internal communication model allows for more control over both ends of the communication. This means that we can make strong assumptions regarding available protocols and algorithms. This also means that, since the software running at each end of the connection originates from the same source, outdated versions are less of a problem assuming a coherent update scheme is easier to manage within the same application. In that model, it is acceptable to update protocols and algorithms much more often than in the external communications model where interoperability and backward compatibility prevents such changes from happening as often as needed.

Existing communication protocols providing secure communication fit well in the external communications model. Although they can be used for internal communication they have non-essential features that provide more potentials attack vectors. In this paper, we focus only on the internal communications model for two reasons. First, most secure communication protocols fit in the external communications model and leave a void for internal communications protocols. The second reason is that the internal communications model gives us more leeway regarding requirements and updates while allowing our secure protocol to be as compact as needed. By considering internal communications as described before we drastically reduce the number of expected situations we have to handle. We have designed the Secure Virtual Connector (SVC) protocol from scratch with the following straightforward objectives: restricting the feature-set as much as possible, reducing the bandwidth overhead incurred by security mechanisms, limiting the number of exchanges needed to establish a secure connection, and providing strong security properties.

In addition to the basic security properties that are confidentiality and authenticity we also consider two additional features that improve the security of exchanges. Making forward secrecy mandatory, we protect data from the potential leak of the keys used to authenticate users. We improve the security even further by ciphering credentials associated to the user so that eavesdroppers have no information about which identity is in use for an exchange.

Table I summarizes the properties provided by three protocols: TLS, SSH, and SVC protocol. We can see that TLS lacks some required security properties, and while SSH provides them, it also provides extra protocol features that are unnecessary and thus reduces network efficiency.

B. Usability requirements

Since our objective is to design a secure protocol in line with modern applications needs for communication, we have to take into account usage constraints. The most important constraints are the mobility of users and also of devices. By considering the restrictions and behaviors of multiple networks and devices we can tailor the SVC protocol to be as efficient as possible without compromising security. Despite network constraints the SVC protocol remains flexible enough to accommodate security policies used for mobile applications.

These constraints appear at different levels. The variety of devices means that an implementation of our protocol must be available on many devices. When designing the protocol, we have carefully choosen data format and security algorithms to ensure that implementations are either readily available or reasonably possible to adapt.

Modern devices are able to switch between a variety of networks to remain connected at any time and locations. While mobile phone networks are mainly used by smartphones they can also operate as gateways to connect any devices in a mobile context. These networks have different capabilities and might even have restrictions on allowed communications. Since secure communications have to reliably cross multiple networks the protocol has to perform all the tasks needed to establish a secure connection with the smallest possible overhead in both quantity of data and number of exchanges.

Security policies regarding mobile applications can cover multiple aspects, including authentication and secure data storage on mobile devices. User mobility brings new challenges regarding reliable authentication of a user on different devices. As an example, for public-key based authentication one can store credentials on the personal computer, but one needs a way to bring them with her/him in a mobile scenario. SVC design takes advantage of the integration between the application and the secure communication protocol. It integrates an application-specific way for authenticating the connection. The protocol expects a digital signature scheme to exist within the application, and delegates the actual authentication process to the application while ensuring that the exchange of credentials is secure.

One last concern is the security of the devices themselves. Different operating systems provide security and cryptographic tools whose behavior can be altered with ease. An example of this is the presence of a system-wide cryptographic keyring. It is mandatory that the security of the protocol remains independent of any OS library or user settings since there can be obsolete libraries or some configuration problems. Such considerations are mandatory and implementations of the SVC protocol must enforce them.

IV. Specification of the SVC protocol

As said before, our SVC protocol fits the internal communications model. It is a lightweight alternative to both TLS and SSH. Functionally, it provides all the properties described in section III with low overhead when compared to non secure communications.

A. Functional design

The SVC protocol operates as an intermediate communication layer between an application and the actual network. The protocol imposes no restriction on the volume of exchanged data. Limitations on underlying algorithms (for example, maximum amount of ciphered data or limited duration of a secure key) are handled appropriately when they are met. SVC works as a transparent packet encapsulation mechanism making the security of the protocol independent of the underlying networks used. The SVC protocol operates independently from specific OS support for security.

We do not consider the need for a large scale PKI as commonly used with TLS, but it is still possible to use one since the protocol does not state which algorithm is used to authenticate the server. The only expectation is that the application can perform a digital signature check on the identity and authentication data exchanged through our protocol.

The negotiation step enforces the following properties: there is a confidential link between the two hosts, client side has authenticated the server side's identity, the server side has authenticated the client side's identity, and the user identity information is kept confidential. Subsequent exchanges take place in the secure context of a virtual connector where data are properly encrypted and protected against alteration. The session key used to protect the confidential link changes regularly in accordance with the constraints of the underlying algorithms used.

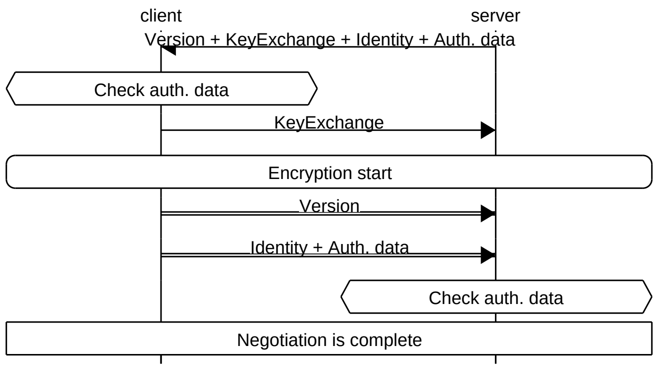

Fig. 1. Secure connection negotiation

double arrows: encrypted content; single arrow: plain-text messages

B. Exchanges summary

The protocol operates in two steps: negotiation, and message exchange. We outline the negotiation process in figure 1. Negotiation confirms the protocol version between the two peers, and starts the key exchange protocol. The server initiates this step once a client has established a connection. Having the first message coming from the server reduces the duration of the negotiation on high latency networks. Client replies only if it can confirm the authenticity of the server. His reply completes the key exchange which is used to initiate the ciphering algorithms. We encrypt the user identity information to avoid unwanted disclosure.

Both the server and client identities can take any form required by the application to perform authentication. Typical identity credentials are a public key, a key identifier, or a certificate. This part is application-specific and the SVC protocol only ensure that this authentication data is securely transmitted. The protocol version serves to identify both the protocol and the algorithms to use for encryption and integrity.

For our first implementation of the SVC protocol, the key exchange relies on Diffie-Hellman and is part of the data that get authenticated. When the key exchange completes, both ends are in possession of a master key named K_M used to generate the first session key named K_1. This key is then used for encryption without needing additional exchanges. The key K_M is also used to generate peripheral information such as additional authentication data.

Following the negotiation phase, each side of the communication can send messages containing actual payload. From the application point of view, both ends only have to deal with the actual payload, and the secure protocol takes care of both encryption and integrity. With both sides keeping track of the amount of data exchanged, when one side hits the limit specified by the current combination of algorithms, it increments the current key identifier and generates a new key K_i. When the other side receives a message with a new key identifier, this identifier become the new active session key, and previous keys are definitely discarded.

C. Algorithms

The cryptographic algorithms used to encrypt the connection, to perform the key exchange, and to handle session keys are specific to the protocol version. This contrasts with usual negotiation steps that allow a selection of algorithms, as this approach led to unsafe combination of otherwise secure algorithms that can have unsafe interactions. By restricting the possible selection to one chain of cryptographic algorithms we ensure that they operate well together.

We put strong restriction on the supported algorithms for two reasons. First, because we have control of both sides of the exchange we know which algorithms are available beforehand. Additionally, we use an in-house portable cryptographic library that include all the latest cryptographic algorithms required. The second reason is that since the SVC protocol operates at the application level, its updates come with a simple application update. Thanks to this updating scheme the choice of algorithms is not an issue.

The first version of the SVC protocol works with the following algorithms:

- Elliptic-Curve based Diffie-Hellman [13] for key exchange
- AES-256 with the GCM mode of operation [14] for encryption
- GHASH function from GCM (which is part of the encryption process) for the Hash-based Message Authentication Code (HMAC)
- SHA-256 [2] for session keys and other secure parameters

When these algorithms (or their combination) become subject to practical vulnerabilities we can specify an updated protocol version using different algorithms. The deployment of this new protocol version relies on the normal update mechanism of the application. Because the entirety of the SVC protocol is at the application level without any dependencies to the OS or to other libraries (as explained in section III-B) updating SVC is not an issue.

Algorithms used to handle public keys and to perform digital signature operations are not mentioned. As explained before, the application provides the digital signature mechanism used to authenticate the exchange.

To allow external authentication mechanism to operate in a secure context, the SVC protocol provides the identity elements to the application on both sides of the communication channel (client and server). The communication protocol manages the confidentiality of the connection and controls when the transfer of identity information occurs to ensure that the user identity remains invisible to an outsider. As a positive side effect, this means that it is possible to integrate our SVC protocol in architectures based on an existing PKI. Even though there are good reasons not to do so, it can facilitate initial adoption of our protocol for existing applications.

D. Keys management

The negotiation produces a session master key through the key exchange algorithm. This master key named K_M is in turn used to produce the session keys used for ciphering.

Session keys generation uses a hash function involving the master key and the session key identifier. For example to produce the first key used for ciphering K_1 we compute $H\left(K_M|1\right)$ where H is the hash function actually in use by the protocol and $|$ is the concatenation of two strings. We compute other security elements that need to be random (like Initialization Vector and the authentication key used for the Hash-based Message Authentication Code) the same way by replacing "1" with other fixed strings. Since we use a cryptographically safe hash function this method produces reasonably independent keys and parameters. These parameters use random elements from both the client and the server as their input reducing the threat posed by a weak or tampered Pseudo-Random Number Generator.

The renewal of the session key triggers on two conditions: after a predefined amount of time, to ensure a limited lifespan of the session key, or after ciphering a certain amount of data, to avoid potential issues with underlying algorithms when keeping the same security parameters for too long [16]. Keeping the same security parameters when ciphering and authenticating too much data can weaken the security of the protocol [16].

E. Networking overhead

The overhead of the SVC protocol when compared to insecure communication is low both in quantity of data and number of exchanges. In normal operation, a single additional round-trip performs the initial negotiation. Each subsequent packet adds almost no overhead: the key identifier is two bytes, and the integrity checking element size is as small as possible while staying secure (see section V for a complete evaluation). Using sensible security parameters it is possible to put all security related data in a 16 bytes block.

The negotiation itself is as compact as possible. The key exchange algorithm relies on elliptic curves arithmetic to reduce the volume of data exchanged while remaining secure. Since the whole negotiation is complete in a single round-trip it is efficient on networks with big latency and is also useful on networks that suffer frequent disconnections.

F. Security analysis

By using well known algorithms and sending sensitive information encrypted we can ensure that the protocol is secure, as it provides confidentiality, integrity, authenticity of both sides, and hides the user identity information. Below is a summary of common attacks that are impossible due to the design properties of the SVC protocol.

Note that when an error occurs (invalid packet, unexpected message, integrity error...) there is no notification of the exact cause of the error. The application sends a single packet with 16 bytes set to zero as part of the error protocol and both ends close the connection. The lack of details about the cause of the error is intentional to avoid potential leak of information. Indeed, indicating failure to validate an authentication tag is useful to forge fake authentication messages [16]. If the application wishes to resume communications after an error it must negotiate the secure connection again to ensure new safe cryptographic keys are generated.

1) Man-in-the-middle: The core of the key exchange protocol is a Diffie-Hellman key exchange [21]. This key exchange is prone to Man In The Middle (MITM) attack. Performing a MITM an attacker can effectively replace the server and communicate with a client through what seems a secure channel. An attacker can also listen transparently on an unsuspecting client to steal information.

The use of digital signature to authenticate the public part of the key exchange prevent this behavior. With the SVC protocol an attacker cannot maintain its position between the client and the server as either he cannot spoof the server identity or cannot decipher the answer from the client.

The worst scenario is a continuous Denial of Service (DoS) attack causing repeated negotiation messages to arrive. An application can detect this behavior (as the whole negotiation process should happen in only one round-trip) and mitigate the issue by throttling the number of secure channel negotiations a single client can make in a limited amount of time.

2) Replay attacks: In secure communications, there is a possibility for an attacker to record a previous negotiation, and then craft a new negotiation request to take the place of either the client or the server. When masquerading as the client a replay attack can impersonate a user. If masquerading as the server an attacker can steal details about the identity of users or more plainly steal whole exchanges.

This is not possible with the way the SVC protocol performs. By having the key exchange as the first step of the negotiation an attacker can never get hold of the session key used. Since both the user identity information and further messages uses encryption and authentication an attacker cannot maintain its position by using a replay attack.

3) Session key retrieval: By using a single session key for a long period, there is an increased risk of leaking information, especially if the attacker has clues about what is actually encrypted. There is also a risk for a brute-force attack: if the same key cipher a large quantity of data, an offline brute-force attack finding that key can be effective.

To avoid this issue the active session key is renewed either because too much time has elapsed or because the same key ciphered too much data. Changing the session key incurs no additional communication and add a negligible computational overhead. Such change can be triggered either by the client or by the server when the key renewal conditions are met, making it practical to protect against session key retrieval attack.

G. Implementation

There are several considerations to take into account for implementation of the SVC protocol. The main consideration is to exclude uncontrolled elements from the secure computations, more particularly unsafe software components. Other considerations exist regarding the availability of the various components required to operate the full protocol.

Excluding uncontrolled elements from secure computations (such as external software libraries) is the primary restriction that improve the security of the SVC protocol over others. When a vulnerability is discovered in a software package, there is a transition period during which the vulnerability is exploitable. This transition period ends with the publication of an update, fixing the issue. When an application depends on third-party software for critical parts of their security, this transition period is extended. In the worse case scenario, there will be no update; this can happen for example when using devices that were made obsoletes by their manufacturer. Such devices are usually used long after this point, making it important to implement critical parts of the software in a way an application developper can control. This consideration is not limited to external software libraries; hardware devices can also be an issue and need to be examinated carefully. Some devices implement hardware acceleration for cryptographic algorithms. Such accelerations usually provide better performances and lower power consumption. Unfortunately, depending on such a component can represent an uncontrolled hazard. In addition, most hardware implementations are hard to audit. Even worse, despite recent advances in countermeasures, using hardware acceleration can led to easier side-channel attacks, a kind of attack that is not as effective on a software implementation. To avoid these pitfalls, the implementation used for SVC needs to be self-contained at the application level.

Another important consideration is the availability of implementations for all required algorithms. New applications have to run on multiple devices that may not share a common hardware architecture. However, these devices shares the same requirements for secure communications. To implement the SVC protocol over multiple devices the underlying algorithms must also be available, with the same requirements as described in the previous paragraph. One solution is to use a portable software library that does not rely on a specific architecture. For example, for our prototype implementation we used a custom cryptographic software library written almost entirely in pure C, and developped within our laboratory. This way we were able to share its functionalities across different devices like desktop computers and smartphones.

By avoiding the use of uncontrolled elements and making sure that the required algorithms are available on any required devices, we were able to transparently implement the SVC protocol for secure exchanges between different type of devices and networks.

V. Evaluation and Examples

A. SVC performances

We consider two examples implying a mobile agent running on a smartphone. First we describe an example where we use Bluetooth communications to exchange data between two smartphones. Then we describe a second example which uses mobile networks to communicate with a remote server over a TCP/IP connection.

These two examples have been chosen to illustrate the compactness of the SVC protocol as both situations have different types of network constraints. In particular their Maximum Transmission Unit (MTU) are different. The MTU is the largest data unit a communication layer can transmit without fragmentation. Messages larger than the MTU can negatively impact the performances of the network. This is especially true for mobile networks where frequent handovers occur (handover is the process of moving communication from one cell station to another). In our case the Bluetooth represents a highly constrained network with low bandwidth and frequent disconnections, while the TCP/IP connection is the *standard* type of connection where bandwidth is usually abundant but might still suffer from connections loss.

We consider digital signatures based on elliptic curves using the $P-521$ curve in the National Institute of Standards Technology (NIST) list of recommended elliptic curves [11]. This curve is also used with the elliptic-curve based Diffie-Hellman to produce keys suitable for the AES-256/GCM encryption. The hash function used for digital signature is SHA-512 to ensure a coherent security level with the elliptic curve used.

In the first example we consider no previous knowledge of both devices identity information. We use Elliptic-curve Digital Signature Algorithm (ECDSA) as the digital signature algorithm. The device that initiated the connection receives the remote public key as the *server identity* (the first communication in figure 1). At this point the users can visually authenticate the received public key in a user friendly way [20] to confirm the remote key. The same happens for the confirmation of the public key on the other device.

After these two verifications the negotiation is complete. Messages are then sent through a secure channel. The public key is roughly 150 bytes long and produces signatures that are approximately 130 bytes long. We also use a simple binary encoding where the length of each variable sized elements is a two bytes unsigned integer. The negotiation requires 440 bytes for each negotiation packet. Because we are using the same authentication process at both ends, the complete negotiation needs 880 bytes of communication and only one round-trip. Bluetooth usually has a default MTU value of 672 bytes meaning that we can send both negotiation packets without stressing the network with fragmentation.

Communications through the secure channel adds 16 bytes of overhead to all exchanged payloads. Key identifier is two bytes. Payload size (before encryption) is represented using a two bytes unsigned integer. The integrity check value in this example uses 12 bytes. This integrity value size is reasonably small while providing a good level of security [16]. Using such a short tag implies regular change of the cipher settings which the SVC protocol do accordingly. The consequence of this is that we can reduce the overhead while staying secure even if the secure connection is used for a long time. Assuming the transmission of packet that fills the default MTU of a Bluetooth network (672 bytes) we have an overhead of 2.4% while providing both confidentiality and authenticity.

The second example uses a mobile network to communicate with a remote server. In the previous example both identities were unknown. We now consider that the client has a list of acceptable server keys, each identified by a single byte integer. During the negotiation process the server sends the appropriate public key identifier and uses the related private key to authenticate its negotiation message. The client then uses the corresponding public key and validates the authentication before sending its own identity in the form of a public key certificate of approximately 1100 bytes. This puts the initial negotiation exchange at 290 bytes from the server and 1390 bytes from the client (for a total of 1680 bytes in one round-trip). We consider a value of 1500 for the MTU of mobile networks (this value is both very common in wired networks and observed in actual mobile networks) and a maximum payload size of 1460 bytes for a TCP/IP segment. In these conditions it is still possible to perform a complete authentication without packet fragmentation making the protocol robust on such networks.

As in the previous example communication through the secure channel has the same static overhead of 16 bytes. Assuming a large quantity of data transmitted that use the maximum packet size of 1460 bytes we have a payload size of 1444 with a security overhead of only 1.4%.

Table II summarizes the network overhead for these two examples. It shows the bandwidth use of security features compared to the values of MTU described before. The last line indicates the bandwidth that remains available for the application payload. In most situations the negotiation step completes without fragmentation making it efficient on networks with high packet loss or frequent need to renegotiate

TABLE II
SECURITY OVERHEAD ON COMMUNICATIONS

Source	Size (in bytes)	% of BT MTU	% of TCP/IP MTU
Negotiation: Both side send public keys			
Server	440	65.5	30.1
Client	440	65.5	30.1
Negotiation: Client authenticate with certificate			
Server	290	43.1	19.9
Client	1390	206.8	95.2
Communications			
Available bandwidth	-	97.6	98.6

a secure channel. Also, the secure communication bandwidth footprint is small meaning that we can use unstable networks securely with good efficiency.

B. Comparison with TLS and SSH

To exhibit the advantages of using SVC over either TLS or SSH, we compared their network footprint in table III. This comparison takes place between the whole SSH negotiation, a partial (but representative) TLS negotiation, and an SVC negotiation. We compare the number of exchanges required, their total size, and the bandwidth available after applying security features. The bandwidth measurements expect a maximum message size of 1460, a common TCP/IP MTU as described before.

SSH evaluation is done using public-key authentication, to closely relate to the SVC test case. It uses AES-128/CTR for encryption and UMAC-64 to authenticate messages.

In the case of TLS, we limited the analysis to the negotiation of a secure connection where only the server is authenticated. Although different (SVC also authenticate the client), it remains relevant since its footprint is larger than the footprint of SVC. In this case, TLS perform a similar Elliptic Curve Diffie-Hellman key exchange (authenticated with a HMAC based on SHA-256), and uses AES-128/GCM for encryption. For convenience there is no client authentication with TLS, but the server is authenticated with its own public key certificate.

In all these comparisons, we can see the overhead incurred by the extra negotiation steps of both SSH and TLS. These steps are clearly redundant in the context of this paper where we consider secure communications between up-to-date, well known entities. The end result is that the SVC protocol provide better performances in such case. Distributed applications are a prime example of such case, because they can be seen as a single "application" running on various nodes. These nodes might be connected using different kind of networks, but the capabilities of each nodes are well known making it redundant to check them every time a secure connection has to be negotiated.

VI. Conclusion and future work

The core of this paper deals with the requirements for real world secure communications regarding security and usability. In addition to the security properties that are confidentiality, authenticity and integrity, we also took into account the capabilities of heterogeneous networks used in modern communications in order to design an efficient secure communication protocol. First, we analyzed the current state of secure communications offered by the two most used protocols that are TLS and SSH. During this analysis we highlighted issues and shortcomings of these protocols. There where also discrepancies between the actual requirements for secure communications in distributed applications and the security provided by these two protocols. Then we analyzed the requirements for internal application communications. As a result we proposed to integrate all the expected security properties in a generic and lightweight protocol, called SVC, well suited for distributed applications over heterogeneous networks. The SVC protocol is a coherent alternative to more general secure protocols when used for internal application communications. It is feature complete and designed to be efficient on different networks. It is a reasonable way to improve the efficiency of secure communications and to reduce the risk of attacks that can take advantage of complex unused features of existing general protocols.

A possible way to extend the protocol is the integration of a secure routing mechanism to take advantage of the encryption and integrity for both the payload and routing headers. Another ongoing development is the use of this protocol to secure communication over a group of clients instead of the basic one-to-one model. Indeed, group communications between distributed processes can be seen as a higher level concept of interaction than the commonly used point-to-point exchange scheme. Primitives for diffusion using group communications can offer several advantages: processes may interact without having explicit knowledge of each other, receivers may be dynamically added or deleted to secure groups, and the activity of a process can be monitored without altering the observed process (this is clearly not possible with the classical *rendezvous* exchange schemes). Moreover, from a theoretical point of view, it appears difficult to encode broadcast in calculi based point-to-point communications, as demonstrated in [8] for a conjecture often expressed before. Therefore, a general theory of secure communicating systems based on multiple exchange instead of *rendezvous* has an intrinsic interest for future networking applications and interactions modelling. Such aspects belong to our future homework and will be presented elsewhere.

TABLE III
BANDWIDTH COST OF DIFFERENT SECURE PROTOCOLS

Protocol	Messages exchanged	Total length (B)	Available MTU bandwidth (%)
SSH (Cipher + Auth.)	16	6383	98.0
TLS (Cipher only)	4	1660	97.5
SVC (Cipher + Auth.)	2	880	98.6

References

[1] CVE-1999-1085 SSH insertion attack, March 2002.
[2] D. Eastlake 3rd and T. Hansen. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF).* Number 6234 in Request for Comments. IETF, May 2011. Published: RFC 6234 (Informational).

[3] Martin R Albrecht, Kenneth G Paterson, and G Watson. Plaintext recovery attacks against SSH. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 16–26. IEEE, 2009.

[4] Jan A Bergstra, Alban Ponse, and Scott A Smolka. *Handbook of process algebra*. Elsevier, 2001.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Number 5280 in Request for Comments. IETF, May 2008. Published: RFC 5280 (Proposed Standard) Updated by RFC 6818.

[6] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. Number 5246 in Request for Comments. IETF, August 2008. Published: RFC 5246 (Proposed Standard) Updated by RFCs 5746, 5878, 6176.

[7] Christian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *Parallel and Distributed Processing Symposium, International*, volume 3, pages 30149b–30149b. IEEE Computer Society, 2001.

[8] Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory*, pages 258–268. Springer, 1999.

[9] Transport Layer Security Working Group and others. The SSL Protocol, Version 3.0. *Netscape Communications*, 1996.

[10] Siddharth Gujrathi. Heartbleed bug: An Openssl heartbeat vulnerability. *International Journal of Computer Science and Engine ter Science and Engineering*, 2(5):61–64, 2014.

[11] Information Technology Laboratory. Digital Signature Standard (DSS). Technical Report NIST FIPS 186-4, National Institute of Standards and Technology, July 2013.

[12] John Marchesini, Sean W Smith, and Meiyuan Zhao. Keyjacking: Risks of the current client-side infrastructure. In *2nd Annual PKI Resarch Workshop*, 2003.

[13] D. McGrew, K. Igoe, and M. Salter. *Fundamental Elliptic Curve Cryptography Algorithms*. Number 6090 in Request for Comments. IETF, February 2011. Published: RFC 6090 (Informational).

[14] D. McGrew and J. Viega. The Galois/Counter mode of operation (GCM). *Submission to NIST.*, 2004.

[15] Microsoft Corporation. Microsoft Security Advisory 2718704, June 2012.

[16] Morris Dworkin. Recommendation for block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical Report SP 800-38D, NIST, November 2007.

[17] Traian Muntean. *Models of parallel computation*. Pergamon Infotech, 1988.

[18] Traian Muntean. *Multi-Models Parallel Programming and Communications*. MIT Press, 1997.

[19] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. 2014.

[20] Adrian Perrig and Dawn Song. Hash visualization: A new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[21] E. Rescorla. *Diffie-Hellman Key Agreement Method*. Number 2631 in Request for Comments. IETF, June 1999. Published: RFC 2631 (Proposed Standard).

[22] E. Rescorla. *HTTP Over TLS*. Number 2818 in Request for Comments. IETF, May 2000. Published: RFC 2818 (Informational) Updated by RFCs 5785, 7230.

[23] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on SSL a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. *Internet: https://www. isecpartners. com/media/106031/ssl_attacks_survey. pdf [June, 2014]*, 2013.

[24] Serge Vaudenay. Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS... In *Advances in Cryptology—EUROCRYPT 2002*, pages 534–545. Springer, 2002.

[25] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Authentication Protocol*. Number 4252 in Request for Comments. IETF, January 2006. Published: RFC 4252 (Proposed Standard).

[26] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Connection Protocol*. Number 4254 in Request for Comments. IETF, January 2006. Published: RFC 4254 (Proposed Standard).

[27] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. Number 4251 in Request for Comments. IETF, January 2006. Published: RFC 4251 (Proposed Standard).

[28] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. Number 4253 in Request for Comments. IETF, January 2006. Published: RFC 4253 (Proposed Standard) Updated by RFC 6668.