Modular Over-The-Wire Security in Managed Publish-Subscribe Systems

Erik Solum, Carl Hauser, and David E. Bakken

Technical Report EECS-GS-010
Washington State University
School of EECS
Pullman, WA 99163

December 2007

Modular Over-The-Wire Security in Managed Publish-Subscribe Systems

Erik Solum
School of EE and CS,
Washington State University
Pullman, USA
erik.solum@gmail.com

Carl Hauser
School of EE and CS,
Washington State University
Pullman, USA
chauser@wsu.edu

David E. Bakken
School of EE and CS,
Washington State University
Pullman, USA
bakken@eecs.wsu.edu

Abstract

The asynchronous publish-subscribe interaction scheme offers many new possibilities for large scale distributed systems. The need for more dynamic information systems in areas like the power grid and other critical infrastructures are well documented. However, widespread use of publish-subscribe systems is held back by the lack of efficient and dynamic security services. In This paper we present a security architecture that takes advantage of a special form of publish-subscribe to provide a modular, over-the-wire configurable, end-to-end security. Addressing critical infrastructures need for high performance and security, while still be able to handle their usually long lifecycle compared to the continuing evolution in the security field.

1 Introduction

The publish-subscribe paradigm (pub-sub) allow for a loose coupling between producers and consumers of information. It leverages a logical asynchronous bus where events can be pushed in at any point by a publisher, and retrieved by any number of subscribers at any other point. This makes it ideal for situations where information production is distributed and is consumed by more than one consumer, such as in many types of information systems for critical infrastructures. Critical infrastructures often have highly distributed sensors that measures the state of the system that a series of control institutions are interested in [10]. A challange with these type of systems is that they often have life expectancies of 25 years or more, and no one can accurately predict the developments in the security field during that time, with respect to how much computational power will be available to attackers and possible breakthroughs in ways to crack specific algorithms. In such environments the security architecture needs to be able to evolve with the changes instead of being at its mercy. Two complicating factors are that the nodes in such systems often are highly distributed and and unmanned, while the data that are sent often are critical and cannot be interrupted.

The widespread use of publish-subscribe systems in this field is held back by the lack of efficiency. The conventional content based pub-sub flavor, where data us dynamically routed based on the content of the event, cannot achieve the same level of performance as the currently used strictly hierarchical systems. This is especially evident when security is added to the equation. *Managed pub-subs* is a flavor of pub-subs that utilizes static routing to achieve the performance that critical infrastructure requires, however little research has been done on providing it with security. This paper will present a dynamic security architecture for managed pub-subs that can address many of the needs currently evident in critical infrastructures such as the power grid. It uses a security management to dynamically assign sets of interchangeable security modules to achieve the end-to-end confidentiality, integrity, obfuscation, authentication and filtering needed to make managed pub-subs a viable alternative for critical infrastructure, while at the same time keeping within real-time data delivery requirements.

The rest of the paper is organized as follows. Section 2 explores both the managed pub-sub model and the threat model, while Section 3 delve into our solution. Section 4 discusses the scalability of the architecture and presents some illustrating experiments, before Section 5 looks into released work. Finally Section 6 concludes the paper.

2 Models

In this section we define the general model for managed pub-sub systems, that Section 3 will extend in order to provide confidentiality, integrity and authentication, and explore the threat model.

2.1 Managed Publish-Subscribe

Managed pub-sub could be considered a compromise between the conventional hierarchical systems and the pub-

sub systems. It supports the asynchronous bus where information can be inserted and extracted anywhere, but instead of routing the events dynamically, a *management plane* is added that control the routing, in what is called the *data plane*. By leveraging static routing, managed pub-sub systems are able to provide the real-time performance needed for critical infrastructure, while at the same time provide convenience of pub-sub systems. An example of such a managed pub-sub system is GridStat[8] that use a hierarchy of quality of service (QoS) brokers to provide the subscribers with QoS guarantees, as seen in Figure 1. GridStat achieves through combination of techniques, such as multicast, filtering, resource control and redundant paths.

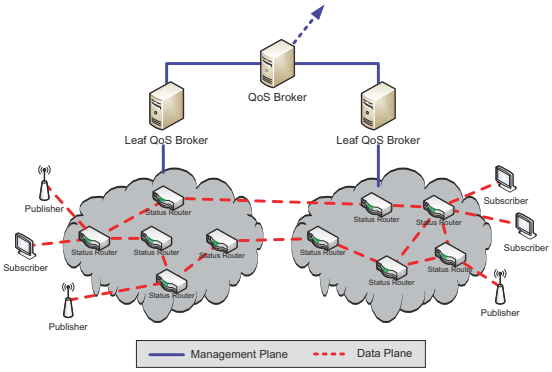

Figure 1. GridStat, a managed pub-sub

Working with security in managed pub-sub makes it possible to make assumptions that are not available when working with other flavors of pub-sub. First of all it is possible to develop end-to-end security, and thus achieve much better performance, since the data does not need to be made available to the routers. Secondly, since there already exist a management it is possible to extend it with security aspects without limiting the systems scalability.

2.2 Threat Model

The threat model is based upon Wang et. al. [18] general security requirements for the pub-sub systems. However Wang et. al. cover a very broad view and include aspects that are not relevant to this problem space such as end-point security and user anonymity. The threat model in this paper has thus a slightly more narrow focus and concentrate on the following application security:

Authentication: Authentication is needed for subscribers to assert that the events they receive actually originated at the correct publisher.

Information integrity: Subscribers needs to be able to check the integrity of received events.

Information confidentiality: Keeping events flowing from publishers to subscribers confidential.

And the following infrastructure security issues:

Publication confidentiality: Assuring that only authorized subscribers gain access to publications.

Service integrity: Protect against attackers mimicking infrastructure level components.

Availability Reduce the risk of malicious publications and subscriptions can be used to overload the system.

It is assumed that the adversary will try to compromise these areas through having access to the network traffic where they can observe, insert, and modify both data plane and management communication.

3 Approach

This section presents the security architecture's dynamic approach to addressing the security of managed pub-sub systems, both with respect to securing the multicast streams of events from the publishers to the subscribers, and the management communication links between the data plane nodes and the management.

3.1 Interchangeable Transparent Software Security Modules

The security architecture is built upon the idea of using transparent interchangeable software security modules to achieve the needed confidentiality, integrity, authentication and to a lesser degree availability. This is accomplished by developing a security extension to the management plane, henceforth called the *security management plane* (SMP), that generate keys and assign sets of modules from a module repository, to the publishers and subscribers, on a per variable granularity according to dynamic policies as illustrated in Figure 2.

Assigning sets of modules and keys on a per variable granularity enables the security system to address the different needs of different multicast streams with status updates, which is called *publications*. For some publications the need for confidentiality might be the strongest concern, and thus be assigned strong encryption modules, others might put emphasis on integrity and obfuscation, while others again might have strict real-time requirements and only use weaker, but faster, security modules. These assignments are specified in *publication policies* and corresponding *subscription policies*.

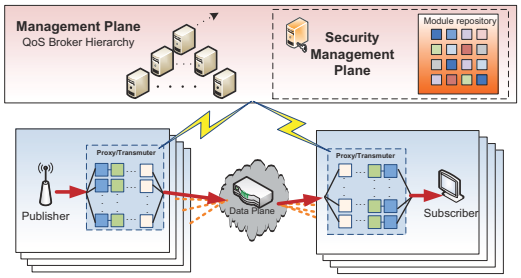

Figure 2. Interchangeable Transparent Modules

Publishers and subscribers gets assigned one publication or subscription policy for each of their respective publications and subscriptions. Based on these policies they download the security modules they need from the security management and instantiate them with the keys specified in the policy. When the modules are installed they are transparently applied to each of the publishers and subscribers event streams.

The SMP's repository of modules can be changed over time by adding new modules at runtime. These new modules can both be assigned to new publications and to existing publications by adding, and/or replacing, old modules assigned to the security groups. All modules added to the module repository need two have two separate parts; a file containing the actual code which will do the modules work and a *module policy* specifying the properties and behavior of the module, and one optional *key generator*, if the module needs a special type of key generations not supported by the default key generator.

The strength of a modular approach is that new security modules can be implemented with varied functionality and performance attributes. By combining the modules in different combinations the security architecture provide a unique toolset for easily making and enforcing tradeoffs between different security and performance properties on an extremely small granularity. It also enables system administrators to respond to changes in the security field by introducing new modules to replace old ones whenever it is needed.

3.1.1 Types of Modules

There are almost an infinite number of modules that could be implemented and deployed, however five major groups of module can be noted as initial interest, each of which has a clear and differentiated goal from the others.

Encryption modules Modules that encrypt information to achieve confidentiality.

Authentication modules Modules that with the use of digital signatures let receivers of the information authenticate its origin.

Integrity modules Error check and error correcting modules which goal is to assert the integrity of the information or correct integrity faults.

Obfuscation modules Modules which goal is to mask recognizable signatures of the data such as repeating patterns, which can be used to break the confidentiality achieved by the encryption module.

Filtering modules Modules that try to reduce the risk of denial of service by filtering published events so only the events that are needed are pushed into the data plane.

By assigning sets of modules from each of these groups to the different publications, the security and performance aspects can be optimized to fit each of their needs.

3.2 Management Communication Security

Adding security measures to provide data plane security, while at the same time introducing new security weaknesses in the management plane of which it relies is futile. Any security system needs to protect its own management communication by providing confidentiality, integrity and accessibility for itself in the same way as it provides its payload, in this case the flow of events from the publishers to the subscribers in the data plane.

The challenges to secure the security management communication are many, but slightly differentiated from the challenges to securing the data plane. Communication in the data plane is based on publishers that push their information trough multicast streams to multiple subscribers with real-time latency requirements. Trough these channels a steady stream of information is transferred, which given enough time, enables an attacker can gather huge amount of data about its security measures.

Security management communication on the other hand is point-to-point communication with sporadic burst of information with relatively loose latency requirements. This gives security management communication much fewer restrictions on what type of security measures that can be taken than with the data plane communication.

Figure 3 depicts the two types of security management plane communication that needs to be secured; the communication from the data plane nodes to the security management plane, represented by a *Security Management Server* (SMS), and the internal security management plane communication between SMSs.

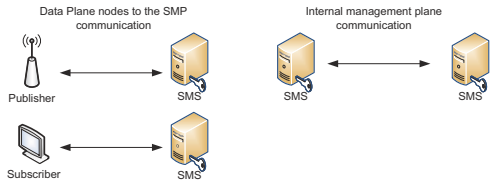

Figure 3. Types of Security Management Communication

The modular approach to securing the data plane necessitates a secure distribution of policies and modules from the security management plane down to the publishers and subscribers. To support such a dynamic solution to the data plane security, a dynamic approach is also needed for the security management communication. A chain is never stronger than the weakest link and building a dynamic security system on top of a static management security would avail little. If some part of the static security is compromised, no level of flexibility in the rest of the system would make any difference.

3.2.1 PKI Shortcomings

Even though PKI has been embraced for Internet security, it is not a silver bullet for every type of security problem. As [9] explores, there are some risks associated with the use of PKI. Most of the risks are related to the PKI implementation and use on the general Internet where thrust issues are raised, such as whether end users really can thrust Certificate Authorities (CAs), but there are also some limitations inherent to the PKI architecture.

PKI security is built upon the assumption that there exists a secure root certificate and, since root certificates must be loaded out of band [1], they cannot easily be updated once installed. Even root certificates with the recommended 4096 bit sized keys do not have eternal lifecycles and needs to be updated. Microsoft recommends replacing a standalone root certificate every 20 years [2]. This is more than long enough for this to be a non-issue for conventional Internet use since users replace their browsers and operating systems at a much faster rate than 20 years, but it poses a problem for systems where out-of-band updates do not have the same rate of update. Information systems for the power grid, for example, are highly distributed with innumerable unmanned devices and a life expectancy well above the recommended root certificate life cycle. There is also no way of guaranteeing that the key is not compromised through unforseen circumstances such as through leaks, whether they are caused by human error or discontent employees. This further emphasizes the need a security architecture that does not rely on only out-of-band replaceable system wide keys.

3.2.2 The pre-load approach

An alternate approach to PKI is to provide all new nodes added to the managed publish-subscribe network, and their parents, with a set of k keys and an initial encryption module, as shown in Figure 4. This makes it possible to build a dynamic solution with some limitations, but which removes the need for static root keys that potentially could be compromised either as results of brute force attacks or human error.

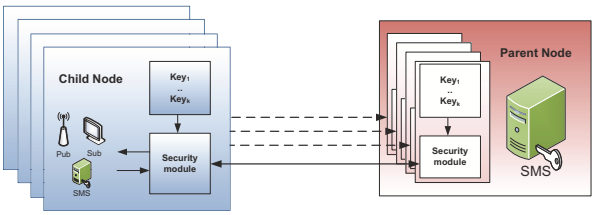

Figure 4. The pre-loading architecture

Pre-loading k keys makes it possible to switch keys k times even with the current key is compromised. This is possible since there is no need to send any keys over the wire, something that makes it impossible for an attacker gain access to the new key through sniffing. The only mischief an attacker can try is to provoke a situation that forces the child and parent node out of key-synchronization. That can either be attempted by initiating false key switches or try interrupting valid ongoing switches.

3.2.3 Security Management Communication re-keying Protocol

Figure 5 illustrates how the parent node initiates a key switch for the management communication link by sending a key-switch command together with a random number encoded with the current key. The child node decrypts the number with the current key, increment the number by one, and returns it together with a new random number encrypted with the next key in the key list. This concludes phase one. If it could be assumed that the child newer would receive false key change commands from attackers masquerading as the parent aimed to push the child out of key synchronization with the real parent, the protocol could have finished here. But since this cannot be assumed the parent node initiates phase two of the protocol which asserts that both sides completed the key change successfully by first checking that the child node correctly incremented the first random number. Then the parent decrypts, increments and re-encrypts the second random number and sends it back to

the child. The child checks that the random number is incremented correctly, increments it a second time and sends it back to the parent. The client now assumes that the key switch is complete and moves permanently to the new key. The server assumes the same when it receives the random number incremented for a second time.

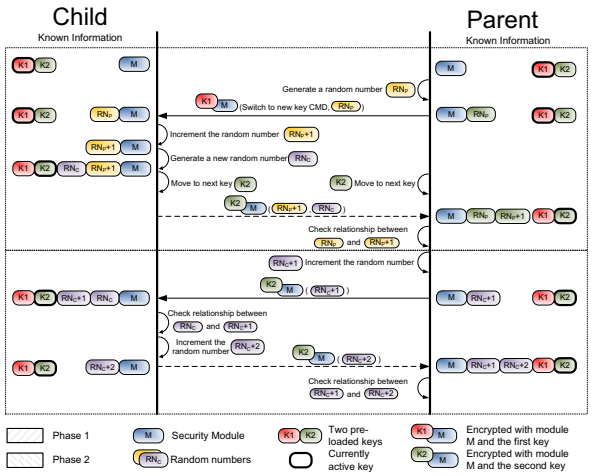

Figure 5. Key switch protocol using pre-loaded keys

Forcing both parties in the key switch to proof their possession of the next key in the list makes it impossible for an attacker to initiate invalid key switches without possessing the next key in the list. If either of the random number checks fails, or either the first message or its reply isn't received, both the child and the parent reverse back to the old key after a short timeout.

To combat the chance that interruptions or loss of the last reply from the child to the parent causing the nodes to go out of key sync, the last message has to be treated differently than the others. Since the child assumes the key switch was successful when it returns the second random number incremented by 2 without checking whether the parent node received it, the parent cannot follow the pattern of the other messages and reset back to the old key if it does not receive it. The parent now has to assert whether the child node has completed the key switch or not.

If the child did not receive the last message from the parent it will reverse to the old key after a given timeout. In an effort to avoid this happening the parent first tries to repeat the last message until he gets a correct response, in which case the key change has been completed successfully, or the time before the child will reverse to the old key runs out.

If the parent has not received any responses, or only invalid ones, when the timeout kicks in, nothing can be asserted about which key the child node uses, the old or the new. To resolve this question the parent initiates a series of *next-key-probes*, as illustrated in Figure 6, with alternating base keys. First the parent sends a probe using the old key as base, if no reply is received, or the reply number is wrong, a new probe using the new key is sent. The parent will continue to do this until it gets a correct reply.

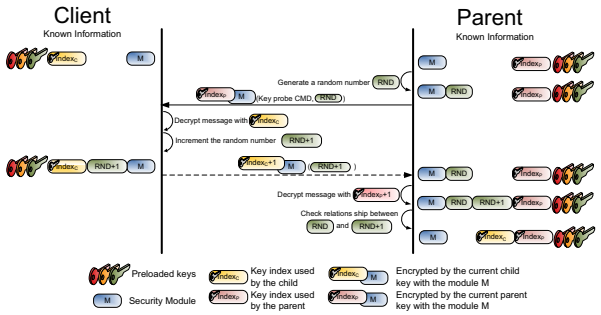

Figure 6. Key Re-synchronization protocol

Since the child has to use either the new key or the old key there can be only two reasons for the parent not to receive a correct response. Either there is network failure, which means that when the network is fixed the probes will re-synchronize the keys, or a man-in-the-middle keeps intercepting the commands. In the case of the man-in-the-middle attack the attacker needs to be able to continuously intercept all the commands between the child and the parent to keep such a denial of service attack up. As soon as the interception stops, the nodes will re-synchronize. Worth nothing is that if the attacker has the control needed over the network needed to accomplish such denial of service it can completely sever the communication between the two nodes without the need to fiddle with specific protocols. Adding additional elaborate measures to combat such attacks on specific protocols would thus be wasteful.

3.2.4 Security Management Communication re-moduling Protocol

Key switches in themselves cannot handle the case where the current key is compromised as a result of the module being compromisable. When the module is compromisable the attacker can by listening in to the message-flow extract the key used. This means that in general all messages that are sent using this encryption module, no matter the number of key switches, are insecure and open to man-in-the-middle attacks after a undeterminable amount of time.

By utilizing the fact that modules in themselves are not secret, it is possible to extend the key switch protocol to also replace the current module, even though it is compromised. Successfully replacing a compromised module with a new module necessitates the transference of the new

module from the parent to the child without any men-in-the-middle compromising the modules integrity. To create a special case where the chance of such a successful man-in-the middle is well below acceptable levels the following assumptions about what preconditions an attacker needs to satisfy to extract a new key from the currently used module have be exploited:

1. Assumption: A relatively large number of processing cycles, which takes time.

2. Assumption: A significant amount of data to base their algorithms on.

These assumptions can be taken advantage of. First of all, by switching keys when sending a new encryption the attacker is denied two things.

1. They cannot use any key they previously have decoded from the communication stream.

2. They have to discard all previously collected data about the stream and start from the beginning.

Since the amount of data needed to transfer a new module is relatively small, the chance of the attacker getting enough data from that transfer alone to decode the new key is very low. Observe that for a man-in-the-middle attack to successfully compromise the integrity of a module transfer the attacker needs to be able to replace the real module with a fake module that it has encrypted with the currently active key. By switching to a new key just before sending the module the attackers workload can be significantly increased by forcing it to extract the new key on very little data before being able to do the encryption. If the attacker by chance should get enough information in that single message with the new module to extract the new key, it still needs some time to calculate it. By adding strict time limits on these transitions the attacker would need to accomplish all this without adding a significant level of latency.

To further enhance the protocol, keys could be switched a second time after the new module is transferred. This will ensure that when the security management communication link starts using the new module, it got a new key that has not been used with the old module. Something that, assuming the new module is secure, makes it impossible for the attacker to use information gleaned from the use of the old module with the new module. Figure 7 depicts the module switch protocol, red arrows symbolize messages encrypted with the old key and module, orange arrows messages encrypted with the temporary key and old module, while the green arrows symbolizes safe communication using the new key and the new module.

First the parent sends a replace module command encrypted with the old key and module. Both the parent and

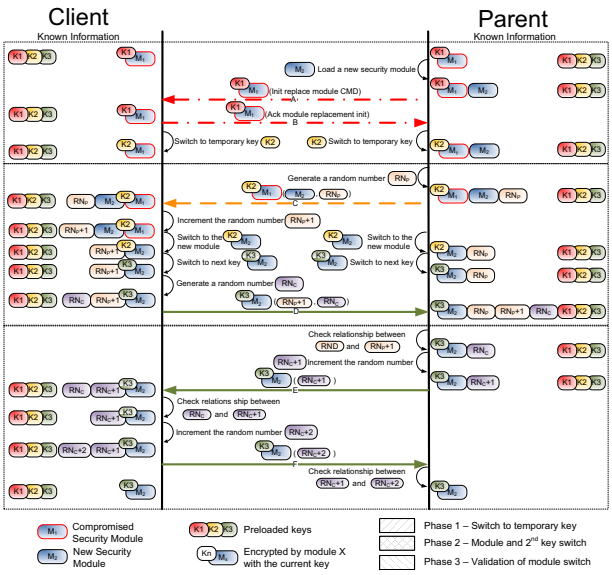

Figure 7. Module switch Protocol using preloaded keys

the child moves to a temporary key. Then the parent sends the new module and a random number, encrypted with the temporary key and the old module. When the child receives message C it decrypts the new module and random number, installs the new module, increment the random number, generates a new random number and then changes keys again to a new key. Finally the child replies to the parent with the two random numbers encrypted with the new module and the new key. Receiving message D the parent also moves from the temporary key to the next key and decrypts the message.

This module switch protocol would make it extremely hard to perform a man in the middle attack and should reduce chance of a successful attack to acceptable levels. The key here is to enforce a tight time requirement on a response to message C. If the parent do not receive a message D within a set time it red-flags the node and aborts the operation. To be able to do a successful man in the middle attack the attacker must be able to extract the temporary key from the singe message C in a short enough time to encrypt its false module with this key and send it to the child without exceeding the time limit.

The third phase in the protocol is added to assert that the module switch was successful and avoid the problem of loss of the last confirming message, caused by an attacker or by mundane network problems, resulting in the child and parent to go out of module and key synchronization. Loss of any intermediate messages can easily be solved by letting both parties to reset to old keys and modules after a time-

out, effectively re-synchronizing the parties. If the parent does not receive F, or the response is invalid, a similar approach to handling the loss of the last message in the key switch protocol illustrated in Figure 6 can be employed to re-synchronize the child and parent.

As shown it is possible to replace a module $k/2$ times if the child is preloaded with k keys. Even though this definitely imposes a finite limitation on the dynamic aspects of the security architecture, it can be argued that with the correct size of k this would not hamper the security significantly. The only reason for needing an infinite number of keys is that the keys and modules keep getting compromised indefinitely. For this to be true there has to be another weakness in the security system that no amount of key or module switches can remedy. Assuming a finite life expectancy there should exist a k such that there are enough keys to switch modules as many times as needed during the deployment of GridStat.

Calculating the size of k is defined outside the scope of this thesis, and would be strictly application dependant, but a simple example is to base the size of k on the expected deployment time divided on the time it takes to replace a module. This would result in the maximum number of keys it is possible to use during the deployment phase of the security architecture's lifecycle. The actual size of k would be much smaller, but it informally proves the point that having a finite k not necessarily makes the security architecture more static.

4 Data Plane Performance

An security extension to, the already mentioned managed pub-sub system, GridStat has been developed based on the presented security architecture. A series of proof-of-concept modules where developed of which many where based on Sun's *Java Cryptographic Extension* (JCE), see Table 1. JCE which has a very limited interface that is not compatible with GridStat's event format. As a result the modules need to employ costly translation logic between the two formats that copies the data at least two times. This means that the numbers presented in Table 1 and 2 has a lot of potential for improvement, but show that even with this room for improvement the security architecture is able to provide security within the real-time requirements.

By employing an end-to-end approach the latency associated with each module becomes constant relative to the length of events routed paths and the added latency they cause can thus be presented without taking the path length into account as shown in Table 1. Since the size of the event obviously affects the number of processor cycles each module needs to perform their task, an experiment is run for each of the modules with a standard integer update and a 200 byte string. The status updates, or events, would in a deployment of GridStat represent readings from sensors and these readings are as a norm primary types such as integers, longs and floats, but in rare cases larger events would be needed and thus supported.

To run the data plane performance experiments a simple GridStat topology was used with a single leaf-QoS Broker, one leaf-SMS, one status router, one publisher, one subscriber and a single publication was set up between the publisher and the subscriber. The whole setup was run on a single machine with a 2.4 GHz Intel E6600 dual-core with 2 gigabytes of RAM running Ubuntu with kernel 2.6.20-16. All GridStat components was compiled and run with java2SE 6 (version 1.6.0-02).

Module	Publisher side latency		Subscriber side latency		End-to-end latency	
	Integer	200 bytes	Integer	200 bytes	Integer	200 bytes
DES[1]	8,059	22,945	9,561	42,486	17,620	65,431
Blowfish[1]	7,336	16,091	9,318	32,263	16,654	48,354
AES[1]	8,341	16,203	8,101	33,231	16442	49,434
TripleDES[1]	9,48	43,642	12,473	70,439	22,221	114,081
AESObfusccation[1]	52,804	57,26	34,134	58,800	86,938	116,060
OneTimePadObf	8,303	34,314	7,489	28,314	15,792	47,959
RSA[1]	$114*10^6$	$114*10^6$	933,402	944,225	$114*10^6$	$114*10^6$
SimpleAuth	2,161	2,189	2,332	2,414	4,492	4,603
MD5ErrorCheck[1]	5,197	7,721	4,815	6,804	10,002	14,525
SHAErrorCheck[1]	6,490	9,823	5,938	9,423	12,428	19,246
SHA512ErrorCheck[1]	11,051	17,698	12,481	19,028	23,532	36,726

Table 1. Proof-of-concept modules and the underlying module logic latencies in nanoseconds

Of the encryption modules, `Blowfish` [17] performed best with `AES` [4] as a close second. The experiments show that both can apply a 128 bit encryption on the stream of integers while adding below 17 microseconds of end-to-end latency and handle the 200 byte stream within 50 microseconds. `DES` [5] and `TripleDES` [6] on the other hand performed much poorer with the larger data samples. DES with only 58 bit effective encryption had an average of 65 microseconds latency on the 200 bytes event and `TripleDES` used as much as 114 microseconds on its status updates. Clearly the `AES` and `Blowfish` modules are the better choice for achieving confidentiality.

The two evaluated obfuscation modules follow two different approaches to achieving obfuscation. The `AESObfuscation` module random generates a new key for each event and uses this key to encrypt it with the AES algorithm, before the random key is appended to the event to allow the receiver to de-obfuscate. As the experiments show this caries a great cost on performance. Since a new key is used for each event the Java virtual machine is unable to optimize the execution of the encryption algorithm and the performance gets much worse than that of the `AES` encryption module that uses the same key each time. This corresponds to the findings done by Opyrchal and Prakash

[1] Modules implemented based on the Suns JCE [19, 3]

[13]. The `OneTimePadObf` module, on the other hand, performs much better, especially with small events. It generates a on-time-pad of equal size to the event and applies it to the event with simple bitwise Caesar Ciphering. After the one-time-pad has been applied it is appended to the event in order to allow the subscribers to reverse the process. The integer experiment shows that data in a event can be completely obfuscated within only 16 microseconds and since the chance of repeating events are greater on smaller data samples one time pad obfuscation seems the more viable obfuscation alternative.

The experiments show that the authentication modules, `RSA` and `SimpleAuth`, based on the asymmetric RSA algorithm and the a static signature scheme respectively, have hugely different performance impacts. The `RSA` module with a 2048 bit key adds as much as 113 milliseconds of latency which is in strict violation with the need for real-time delivery of information, while the `SimpleAuth`, because of its simplistic logic only adds 4.5 microseconds in the 200 bytes experiment. `SimpleAuth` is an alternative to using resource expensive asymmetric algorithms by simply adding a set of secret identification bytes to each event in order to sign them. Though such an approach in itself would achieve little, since an attacker easily could replicate such a signature, combined with encryption and obfuscation modules it can achieve much of the same effect that using asymmetrical algorithms can provide, but with the fraction of the cost. Adding a secret signature to each update, and then apply an obfuscation module that randomizes the signature before it is encrypted by a third module, would make it extremely difficult for an attacker to inject falsified information. First of all it would need to know the secret signature; secondly it would need to know the key used by the encryption.

The three integrity modules implemented and evaluated, are error checking modules that uses different hash algorithms to assert the integrity of reviewed information. The `MD5ErrorCheck` [16] module generates a 128 bit hash, the `SHAErrorCheck` [7] uses 160 bit hash and the `SHA512ErrorCheck` module employs a 512 bit hash of to achieve a varying degree of integrity checks. The varying length of the hash impacts the added end-to-end delay, but they scale well from the integer size experiments to the 200 bytes experiment and show that a 200 byte sized events on average can be error checked with either a MD5 hash or a SHA hash within 20 micro seconds end-to-end.

To test how the performance was affected by combining modules into sets the experiments in presented in Table 2 was undertaken. As the two first experiments show the order of which modules are organized affect the latency they add. This is due to the different inflation of the data size, e.g. applying a hash module before applying an encryption module will require the encryption module to encrypt the hash and thus increase the latency.

#	Module set	Publisher side latency		Subscriber side latency		End-to-end latency	
		Integer	200 bytes	Integer	200 bytes	Integer	200 bytes
1	MD5ErrorCheck Blowfish	16,612	27,208	14,736	35,231	31,348	62,439
2	Blowfish MD5ErrorCheck	12,746	24,588	10,412	34,434	23,158	59,022
3	OneTimePadObf Blowfish MD5ErrorCheck	14,042	54,404	14,736	52,885	28,778	107,289
4	OneTimePadObf AES SHA	15,376	60,701	15,946	60111	31322	120813
5	SimpleAuth OneTimePadObf Blowfish	13,393	46,717	11,392	39115	24785	85833
6	AESObfuscation TripleDES SHA512	70,602	135,048	59,304	108,073	129,906	243,121
7	SimpleAuth OneTimePadObf Blowfish SHA	15,765	59,583	16,577	49,986	32,342	109,569

Table 2. Module set latencies in nanoseconds

Experiment 3 and 4 in Table 2 show that combinations of one-time-pad obfuscation, Blowfish/AES encryption and MD5/SHA integrity modules can be applied to a stream of integers events with an average added latency of below 32 microseconds and 120 microseconds for the 200 byte stream. Experiment 5, on the other hand, show that if we instead of error checking is interested in authenticating the stream the added latency can be reduced down to 85 microseconds for the 200 bytes stream.

To show a worst case scenario experiment 6 combines the three slowest modules non-asymmetric modules; `AESObfuscation`, `TripleDES` and the `SHA512-ErrorCheck`. As a result the added latency gets as high as 130 microseconds for integers and 243 for the 200 bytes. This clearly not a good combination of modules, but shows that as long as the latency is constant related to the length of the event stream even this combination can be tolerated.

Experiment 7 takes it one step further and show that we are able to provide authentication, obfuscation, encryption and integrity by combining the `SimpleAuth`, `OneTimePadObf`, `Blowfish` and `SHAErrorCheck` module while just adding 32 microseconds of latency for the integer stream and 110 for the 200 bytes stream. This clearly shows that even with the non-optimized proof-of-concept modules evaluated here the modular approach is able to provide the security needed for the event streams while adhering to the stated real-time requirements. Future optimization will only further reduce the added latency and enable even higher levels of encryption.

5 Security Management Scalability

In order to make the presented security architecture scalable the management plane has to implement some type of distributed topology. While there are different ways of which this can be accomplished, the DNS [12] approach to a hierarchical structure has proven to support both scale and efficiency. A proof-of-concept management plane based on this organization was thus implemented as shown in Figure 8.

The security management was divided into leaf- and interior-SMSs. Each leaf-SMS controls the publication policies in their own subset of the data plane, in addition to all subscription policies, local or global, associated with these publication policies. The interior-SMSs routes commands between global subscribers' leaf-SMS and the leaf-SMS controlling the corresponding publication policy.

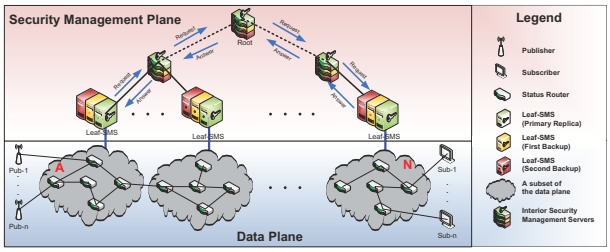

Figure 8. Security Management Organization

By keeping all the access control at the leaf level two challenges are addressed. First of all the computational work is kept in the leaf nodes and thus not impacted by the scale of the infrastructure topology of which it is part of. Increasing the number of levels in the hierarchy will therefore only add a linear latency increase caused by additional routing hops.

Secondly it addresses that many large distributed information systems spans security domains. Since all access control to published information is done at the local leaf-SMS, the local security domain will be in complete control of their own information and thus ease industry concerns about this issue.

5.1 Experiments

The following experiments where undertaken to show how scaling of the security architecture impacted its performance. All experiments where run on four nodes with a Intel 2.4 Ghz E6600 dual core CPU and 2 gigabytes of RAM. The nodes ran Linux Fedora 6 with kernel 2.6.22.7-57 and java2SE 6 version 1.6.0-02 on a 100 Mbit Switched LAN. All the security management communication lines are secured with a 128 bit AES[4] module.

The experiments measures the time it takes to replace a policy being used by a publication and a set of subscribers with a new policy. The operation consists of the leaf-SMS that initiate the changes completing the following tasks:

- Update its publication and subscription policy data base so it adheres to the new SGP.
- Push out the new publication and subscription policies to the affected publishers and subscribers.

And the publisher and subscribers that receives the policy update activating the new policy by:

- Downloading the new module(s) if they do not already posses it/them.
- Instantiating and activate the new module set with the new set of keys.

The first set of experiments focused on the number of levels in the security hierarchy and one through five levels of SMSs was tested.The experiments show, as Figure 9 illustrates, that there are a linear increase in the latency. If the the SMSs where allowed to cache the modules the latency increased from 136 milliseconds with one level in the hierarchy, to 162, with 5 levels in the hierarchy, when assigning a new 5417 Blowfish[17] module and corresponding key.

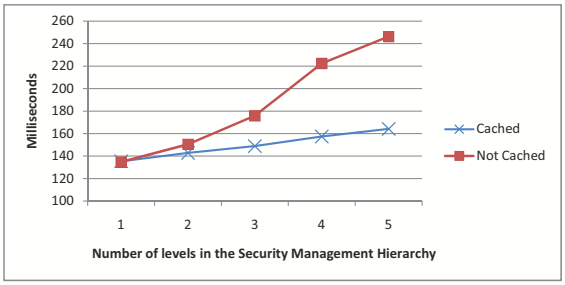

Figure 9. Latency relative to number of levels in the Hierarchy

The next set of experiments focused on the impact of the number of subscribers when the module cache is enabled. The same setup using a five level security management hierarchy was used, but instead of a single publisher and subscriber, the number of subscribers where increased up to seven. Figure 10 show the latencies decomposed into its three main components.

As we can see from the decomposition, the time it takes to update the policy database and the time it takes to download the modules stays fairly constant relative to the number of subscribers, while the time it takes to push out the policies have a linear increase. The downloading and activation

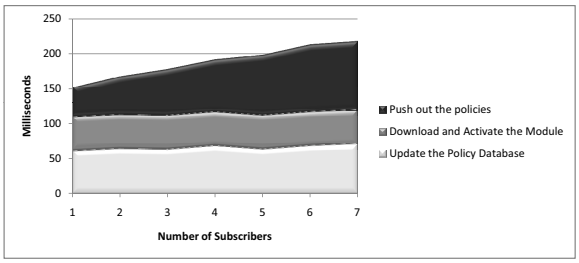

Figure 10. Latency relative to number of subscriptions

of the modules are kept constant because the subscribers do this in parallel.

6 Related Work

The publish-subscribe research tends to focus on performance, scalability and expressiveness [18] and the research on security related matters that is being done is mainly concentrated on *content-based* publish subscribe (CBPS)[14, 13, 15, 11]. In CBPS systems the subscribers registers filter functions and events are routed based on evaluations of these functions applied on the, usually complexly structured, events. Since the brokers in a CBPS require full or partial knowledge of the content of the events to route correctly the introduction of confidentiality poses interesting research questions. Most approaches published on how to address these requirements have been different variations on encrypting each attribute of an event with different keys, and thus allow brokers with different needs to decrypt just what is needed to route the event successfully [13]. Khurana [11] presents an extension to this approach that supports events that have an XML structure. Others employ end-to-end techniques such as only encrypting the values of events, while keeping the attribute types in clear text to route the information [15].

While most research on security in CBPS is not directly applicable to managed publish-subscribe, some of it can be useful such as Wang et. al. [18] which does not present a security architecture, but explores the issues related to security in the publish-subscribe paradigm that can be used as part of the threat model. Pesonon et. al. [14] present a security architecture for multi-domain CBPSs that acknowledges the fact that confidentiality in multi-domains warrant special considerations such as to what the intermediate brokers are allowed to know about the content of routed events and where the access control to the events should be placed. The architecture employs two separate security schemes, one for the connection between the end points (publishers and subscribers) to the brokers based on TLS and another for the inter-broker communication that allows the brokers to access the events partially.

A common trait of the security architectures published for CBPS is that they all employ PKI in a lesser or greater fashion. This introduces weaknesses that are explored in Section 3.2.1.

7 Conclusions

In this paper we have presented a security architecture that through the use of transparent interchangeable software modules can ensure confidentiality, integrity, obfuscation, authentication and filtering to managed pub-sub systems, while adhering to critical infrastructures' real-time requirements. That enables the security system to provide the needed application security defined in the threat module. It also improves availability by filtering misbehaving publishers. The architectures support for secure re-keying and re-moduling of both the data plane and the management communication without relying on root-certificates, provide service integrity without locking the system to a single set of security algorithms, while the leaf-SMSs provide publication confidentiality by only allowing authenticated subscribers with the right credentials to access the needed modules and keys to read the event streams.

The scalability analysis and the experiments showed that the architecture can support large scales and still provide the necessary performance while keeping all data access control local to each security domain. All this makes the use of the presented security architecture a ideal tool for providing pub-subs with the security they need to be utilized in critical infrastructures.

8 Acknowledgements

This work was funded in part by the U.S. Department of Energy, Office of Electricity Delivery, via subcontract 49944 with Pacific Northwest National Laboratory, and the CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) grant.

References

[1] ITU Telecommunication Standardization Sector (ITI-T). ITU-T RECOMMENDATION X.509, July 2005, URL: http://www.itu.int/rec/T-REC-X.509-200508-I/en.

[2] Microsoft TechNet. Certificate Life Cycle, http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/distrib/dscj_mcs_evyy.mspx ?mfr=true Last Checked: 2007-10-02.

[3] Sun Microsystems. Java SE Security, Sun Developer Network URL: http://java.sun.com/javase/technologies/security Last checed 10.25.07.

[4] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). Federal Information Processing Standards Publication (FIPS PUB) 197, Nov 2001.

[5] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Federal Information Processing Standards Publication (FIPS PUB) 46-3, October 1999.

[6] National Institute of Standards and Technology (NIST). Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. NIST Special Publication (NIST SP) 800-67, May 2004.

[7] National Institute of Standards and Technology (NIST). Secure Hash Signature Standard (SHS). Federal Information Processing Standards Publication (FIPS PUB) 180-2, August 2002.

[8] D. E. Bakken, C. H. Hauser, H. Gjermundrod, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical Report 009, Washington State University, May 2007.

[9] C. Ellison and B. Snhneier. Ten risks of pki: What youre not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[10] C. H. Hauser, D. E. Bakken, and A. Bose. A failure to communicate: next generation communication requirements, technologies, and architecture for the electric power grid. *Power and Energy Magazine, IEEE*, 3(2):47–55, April 2005.

[11] H. Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 801–807. ACM, 2005.

[12] P. Mockapetris. RFC 1035: Domain names - implementation and specification, November 1987. URL: http://tools.ietf.org/html/rfc1035.

[13] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 21–21, Berkeley, CA, USA, 2001. USENIX Association.

[14] L. I. W. Pesonen, D. M. Eyers, and J. Bacon. Encryption-enforced access control in dynamic multi-domain publish/subscribe networks. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 104–115, New York, NY, USA, 2007. ACM Press.

[15] C. Raiciu and D. Rosenblum. Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures. In *Proc. Second IEEE Communications Society/CreateNet Int. Conf. on Security and Privacy in Communication Networks (SecureComm 2006)*, Aug.-Sep. 2006.

[16] R. Rivest. RFC 1321: The md5 message-digest algorithm, April 1992. MIT Laboratory for Computer Science and RSA Data Security, Inc.

[17] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, December.

[18] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 303, Washington, DC, USA, 2002. IEEE Computer Society.

[19] J. Weiss. *Java Cryptography Extensions: Practical Guide for Programmers*. Morgan Kaufmann, 1 edition edition, 2004.