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

Erik Solum, Carl Hauser, David E. Bakken, Rasika Chakravarthy
School of EECS
Washington State University
Pullman, USA
erik.solum@gmail.com, chauser@wsu.edu, bakken@eecs.wsu.edu, rmchakra@wsu.edu

Abstract

This paper presents a modular, software-based, over-the-wire configurable, end-to-end security architecture for managed publish-subscribe systems. Intended for use in critical infrastructure systems, the architecture provides mechanisms allowing it to evolve, during operation, over the long lifetimes typically encountered in these systems by allowing security modules to be securely added and replaced at runtime. Asynchronous publish-subscribe schemes offer many new possibilities for communication in large-scale distributed systems but their use has been held back by lack of efficient and dynamic security architectures. Our security architecture addresses these needs with dynamically replaceable security modules for confidentiality, integrity, authentication, and obfuscation. The variety of available modules provides tradeoffs between performance and security now and for the future. Experimental performance results for the various existing modules, in the context of the architecture, are presented.

1 Introduction

The publish-subscribe paradigm (pub-sub) allows loose coupling between producers and consumers of information. It implements a logical, asynchronous bus where events can be pushed in at any point by a publisher, and retrieved by any number of subscribers at other points. This makes it ideal for situations where information production is distributed and more than one consumer is interested in each information item, such as in information systems for monitoring and controlling large-scale critical infrastructures. Critical infrastructures often have highly distributed sensors that measure the state of the system and report their data to several control institutions [12]. A challenge with securing this type of systems is that they often have life expectancies of 25 years

or more, yet 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-related components must evolve along with cryptographic developments instead of being at their mercy. Two complicating factors are that the nodes in such systems are often widely distributed and unmanned, while the data that they produce are critical and their flow cannot be interrupted.

The widespread use of pub-sub systems in this field is held back by the lack of efficiency. Conventional content-based publish-subscribe, in which data are dynamically routed based on the content of the event, cannot achieve the same level of performance as the currently used strictly hierarchical (SCADA) systems. This is especially evident when security is added to the equation. *Managed publish-subscribe* [11] uses static routes configured by a management plane, to achieve the performance that critical infrastructures require. However a security solution is also needed. We present a dynamic security architecture for managed publish-subscribe systems that addresses many of the data exchange and related communication needs currently evident in critical infrastructures such as the power grid. We introduce a *security management plane* to dynamically assign sets of interchangeable security modules in order to provide end-to-end confidentiality, integrity, obfuscation, authentication and filtering, while at the same time keeping within real-time latency requirements for data delivery.

The rest of the paper is organized as follows: Section 2 explores both the managed pub-sub model and the threat model; Section 3 presents our architecture; Section 4 discusses the performance of our solution using proof-of-concept modules; Section 5 looks into related work and Section 6 concludes the paper.

2 Models

In this section we define the system model for managed pub-sub systems and we describe a threat model, against which our modular security system protects.

2.1 Managed Publish-Subscribe

Managed pub-sub could be considered a compromise between the conventional hierarchical systems and content-based pub-sub systems. The managed pub-sub model provides an asynchronous data bus where information can be inserted and extracted anywhere, but instead of routing the events dynamically based on their content, a *management plane* is added that statically controls the routing based on publication identities. By leveraging static routing, managed pub-sub systems are able to provide the real-time performance needed for critical infrastructure monitoring and control, while at the same time providing the convenience of pub-sub systems. GridStat[8] is an example of such a managed pub-sub system. It uses a hierarchy of quality-of-service (QoS) brokers to provide QoS guarantees, as seen in Figure 1. GridStat achieves its goals through a combination of techniques, such as multicast, filtering, resource control and redundant paths. Managed pub-sub makes possible several assumptions when developing a security architecture that are not true in other varieties of pub-sub. 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 forwarding engines. The existing management infrastructure can be extended to support security

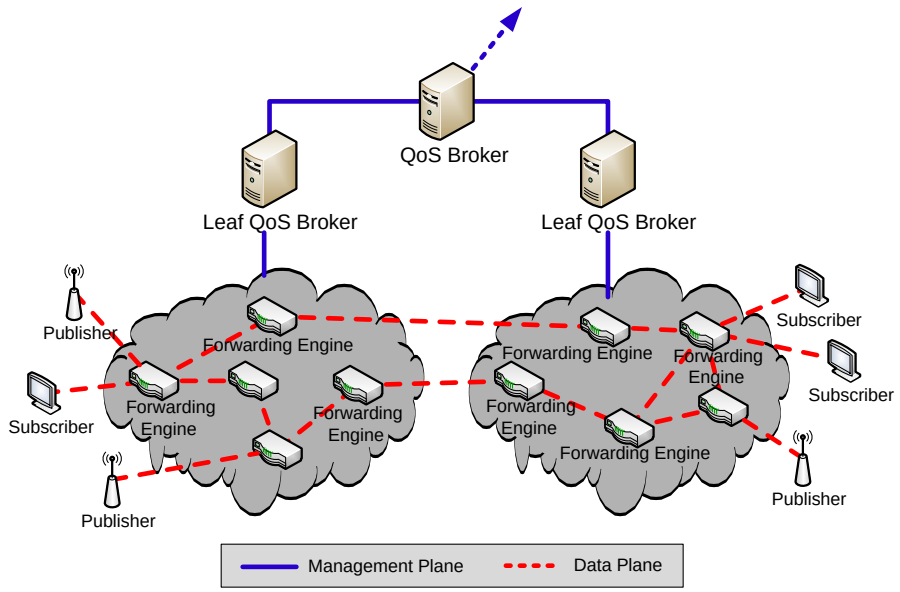

Figure 1. GridStat, a managed pub-sub

without further limiting the system's scalability.

2.2 Threat Model

The threat model is based upon Wang et al. [19] general security requirements for pub-sub systems. The threat model addressed here, has a slightly narrower focus and concentrates on the following aspects of application security:

authentication: subscribers must be able to determine that the events they receive originated at the intended publisher

information integrity: subscribers must be able to check the integrity of received events

information confidentiality: events flowing from publishers to subscribers must be kept secret from the pub-sub infrastructure and from adversaries.

and on the following infrastructure security issues:

publication confidentiality: publishers must be assured that only authorized subscribers gain access to publications

service integrity: protect against attackers mimicking infrastructure level components

availability: reduce the risk that malicious publications and subscriptions can be used to overload the system.

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

3 Approach

The security architecture uses a dynamic approach to addressing the security of managed pub-sub systems. This approach has been applied to secure the multicast streams of events from the publishers to the subscribers i.e, the publications, and also to secure the management communication links between the data plane nodes and the management nodes.

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. A security extension to the management plane, henceforth called the *security management plane* (SMP), generates keys and assign sets of modules from a module repository to publishers and subscribers, on a per-flow granularity according to dynamic policies as illustrated in Figure 2.

Assigning sets of modules and keys on a per-stream granularity enables the security system to address the different needs of each different multicast stream (called a *publication*). For some publications the need for confidentiality might be the strongest concern, requiring strong encryption modules; others might emphasise integrity and obfuscation; yet others might have strict real-time requirements so that weaker, but faster, security modules are needed. These requirements are specified in *publication policies* and corresponding *subscription policies*.

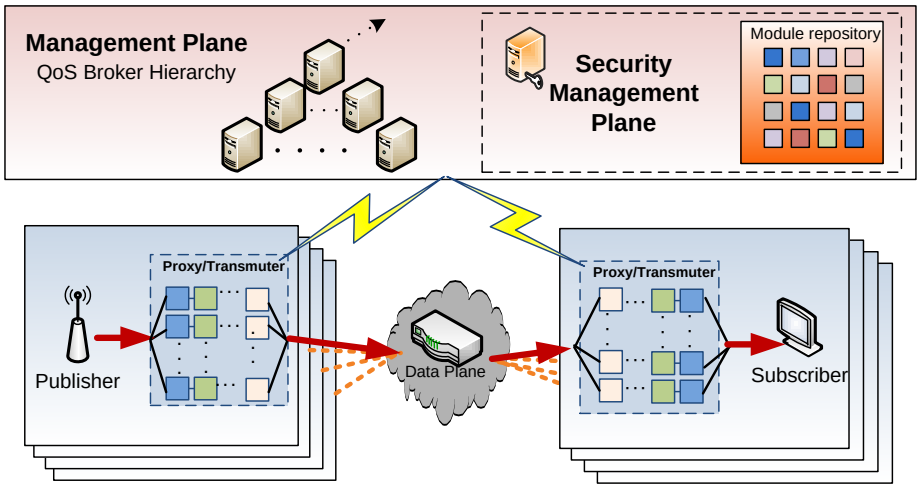

Figure 2. Interchangeable Transparent Modules

Publishers and subscribers are assigned one publication or subscription policy for each of their respective publications and subscriptions. Based on these policies they download the security modules that they need from a security management server and instantiate them with the keys specified in the policy. Once the modules are installed they are transparently applied to each of the publisher's or subscriber's event streams.

The SMP's repository of modules can be changed over time by adding new modules at runtime. These new modules can be assigned both to new and to existing publications through adjusting

their policies stored in the management plane. All modules added to the module repository need to have two separate parts: a file containing the actual code which does the module's work and a *module policy* specifying the properties and behavior of the module. An optional third component, a *key generator*, is included if the module needs a special type of key generation 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 allowing modules to be combined in different ways the security architecture provides a toolset for easily making and enforcing tradeoffs between different security and performance properties at a small granularity. It also enables system administrators to respond to changes in the security field by introducing new modules to replace old ones whenever necessary.

3.1.1 Types of Modules

There are almost an infinite number of modules that could be implemented and deployed, however five major kinds of modules are of 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 use digital signatures to let receivers of the information authenticate its origin

integrity modules: Error checking and error correcting modules whose goal is to assert the integrity of the information or correct integrity faults

obfuscation modules: Modules whose goal is to mask recognizable patterns in data that could 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 that only 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 mechanisms to provide data plane security, while at the same time introducing new security weaknesses in the management plane on which it relies would be futile. Every security system needs to protect its own management communication by providing confidentiality, integrity and accessibility for itself in the same way as it provides it for its payload.

Communication in the data plane is based on publishers that push their information through multicast streams to multiple subscribers with real-time latency requirements. These channels carry a steady stream of information, so given enough time, an attacker can gather a huge amount of data about its security measures. Security management communication, on the other hand,

is point-to-point and consists of sporadic bursts of information with relatively loose latency requirements. Security management communication thus suffers fewer restrictions on what types of security measures can be taken.

There are two types of security management plane communications that need 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.

The modular approach to securing the data plane necessitates a secure distribution mechanism for policies and modules from the security management plane down to the publishers and subscribers. To support a dynamic solution to data plane security, a dynamic approach is also needed for security management communication. Building a dynamic data plane security system on top of a static security management system would avail little because if some part of the static security management system were to be compromised, no level of flexibility in the rest of the system would make any difference.

3.2.1 The PKI approach

Even though public-key infrastructures (PKIs) have been embraced for Internet security, they cannot solve every type of security problem. As [10] explores, there are risks associated with use of PKI. Most of the risks are related to the PKI implementation and its use on the general Internet where trust issues are raised. For example, can end users really trust Certificate Authorities? There are also some limitations inherent in 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 need 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 long before they reach 20 years of age. The recommendation poses a significant problem, however, for systems where out-of-band updates are difficult and infrequent. Information systems for the power grid, for example, are highly distributed with large numbers of unmanned devices and a life expectancy well above the recommended root certificate life cycle. There is also no way of guaranteeing that a root certificate key remains uncompromised for its expected lifetime, whether the failure is caused by human error, malice, or system error. This further emphasizes the need for a security architecture that does not rely on system-wide keys that are only replaceable out-of-band.

3.2.2 The pre-loaded keys and modules 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 3. The parent shares a different set of k keys with each of its child nodes. 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 a result of brute force attacks or errors.

Pre-loading k keys makes it possible to switch keys k times even if the current key is compromised. This is possible since there is no need to send any keys over the wire, hence no possibility

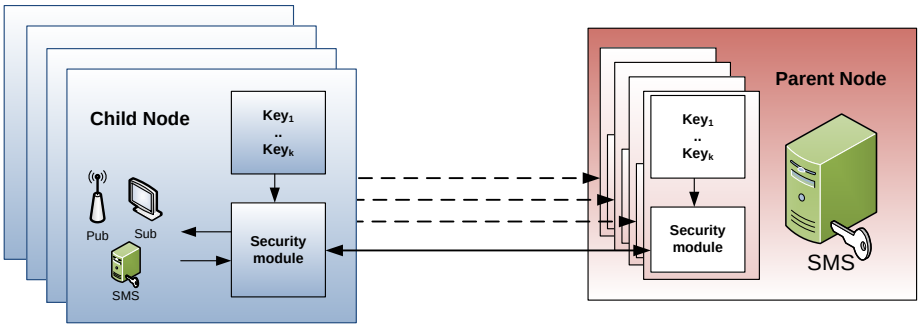

Figure 3. The pre-loading architecture

for an attacker to 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 attempting to interrupt valid ongoing switches. These possibilities are addressed by the re-keying protocol.

Pre-loading k keys out of band is an expensive operation and it should ideally be done only once during the lifetime of each node. Adopting the standard practice of generating session keys for protecting the communication not only preserves the pre-loaded keys for longer, but also increases the number of keys dynamically.

3.2.3 Security Management Communication Re-keying Protocol

Figure 4 illustrates how a 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, increments 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 never would receive false key change commands from attackers masquerading as the parent trying to push the child out of key synchronization with the real parent, the protocol could finish 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 child now assumes that the key switch is complete and moves permanently to the new key. The parent assumes the same when it receives the random number incremented for a second time.

Forcing both parties in the key switch to prove 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 if either the first message or its reply isn't received, both the child and the parent revert 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 causes 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

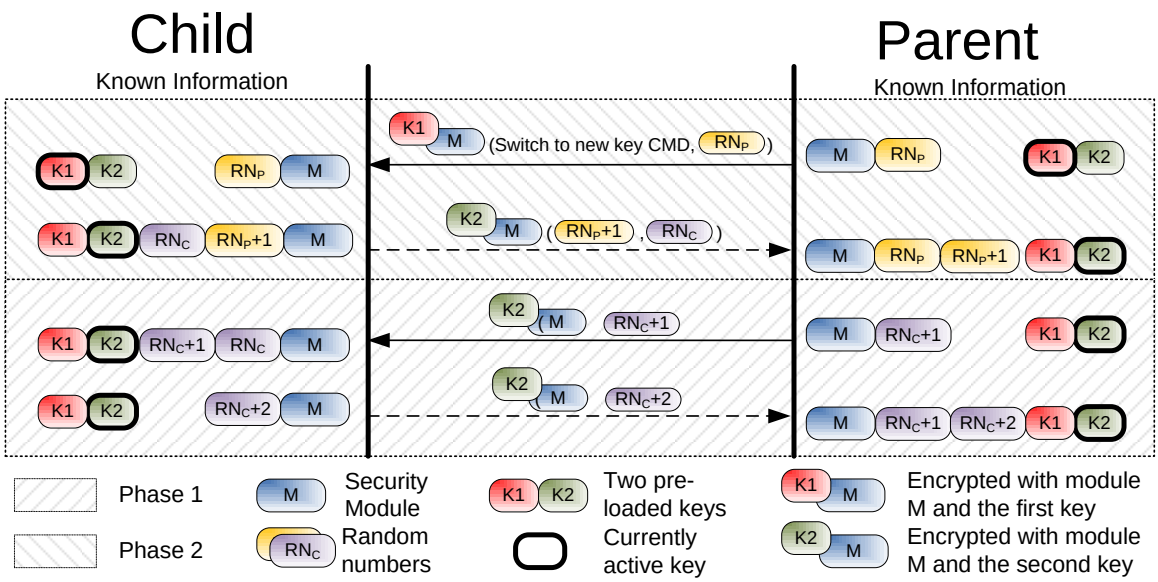

Figure 4. Key switch protocol using pre-loaded keys

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 determine 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 revert to the old key after a given timeout. During this timeout the parent will try to repeat the last message in an effort to successfully complete the key change. 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 5, 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.

Since the child has to use either the new key or the old key there can be only two reasons for the parent not receiving a correct response. Either there is a 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 noting is that if the attacker has enough control over the network to accomplish such denial of service it can completely sever the communication between the two nodes without the need to attack specific protocols. Adding additional elaborate measures to combat such attacks on specific protocols would thus be useless.

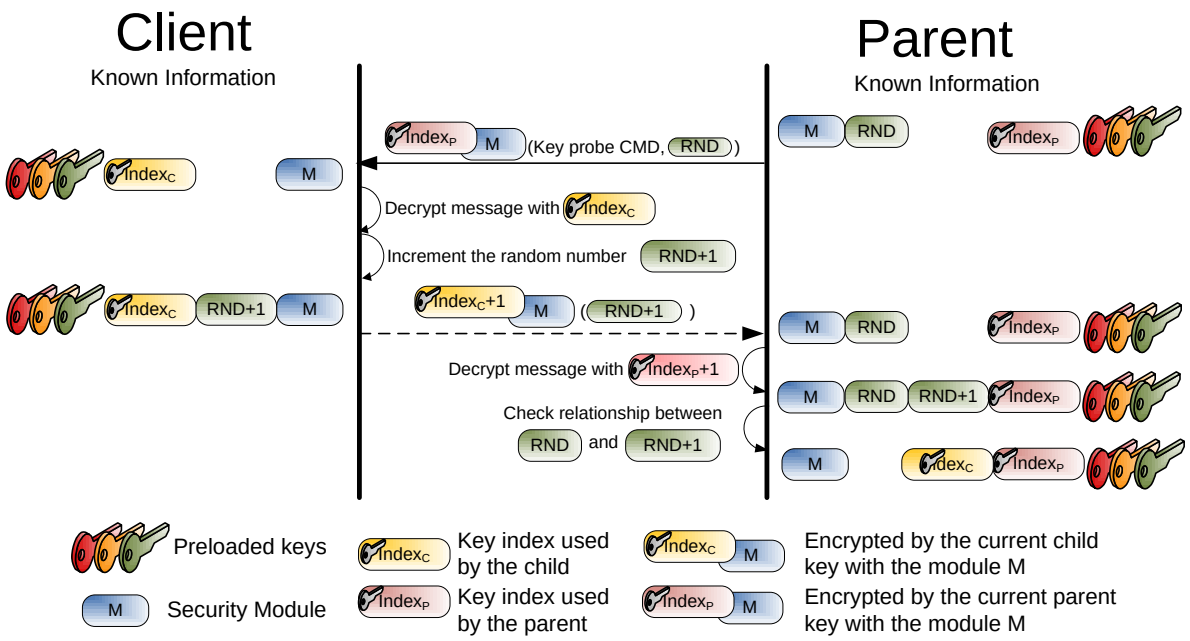

Figure 5. Key re-synchronization protocol

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 current module being compromisable. When a security module is compromisable the attacker can, by listening 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 some 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 new module's integrity. To create a special case where the chance of such a successful man-in-the middle attack 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 been exploited:

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

2. Assumption: A significant amount of data encrypted with a given key.

These assumptions are used as follows. First, by switching keys when sending a new encryption module the attacker is denied two things:

1. It cannot use any key it previously may have decoded from the communication stream.

2. It has 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 attacker's 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 gets 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 should 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 has a new key that has not been used with the old module. Assuming the new module is secure this again makes it impossible for the attacker to use information gleaned from the use of the old module with the new module. Figure 6 depicts the module switch protocol, arrows with lines and dots symbolize messages encrypted with the old key and module, stippled arrows symbolize messages encrypted with the temporary key and old module, and whole arrows symbolize safe communication using the new key and the new module.

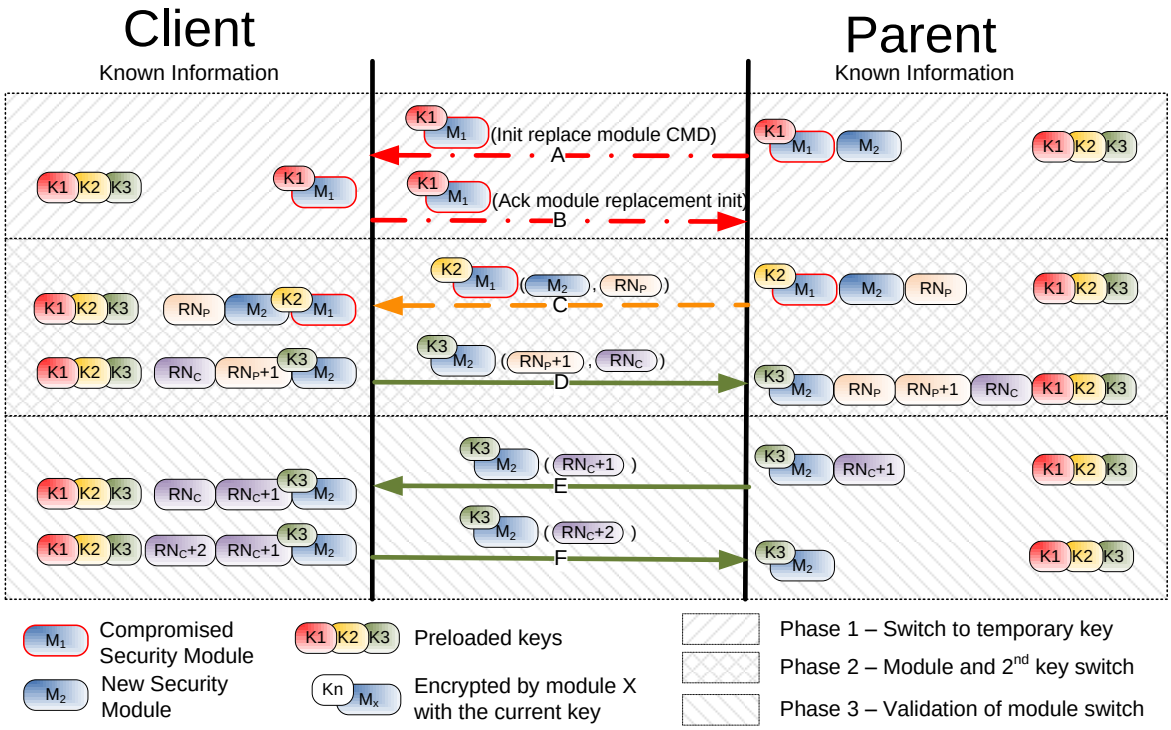

Figure 6. Module switch protocol using pre-loaded keys

The protocol in detail is: first the parent sends a replace module command encrypted with the

old key and module. Both the parent and the child move to the next key in the list, which becomes 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, increments 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 does not receive 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 single 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 going out of module and key synchronization. Loss of any intermediate messages can easily be solved by letting both parties reset to old keys and modules after a timeout, 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 5 can be employed to re-synchronize the child and parent.

As shown it is possible to replace a module $k/2$ times over the life of the child, 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.

An informal way of showing that the number of keys needed to be preloaded is finitely bounded is to divide the system's projected time of deployment by the time it takes to do one round of re-keying. While this number in most cases would be impractically large, environment and statistical analysis can be used to reduce the number to a more manageable level. Calculating an appropriate k is application dependent and is outside the scope of this paper.

4 Data Plane Performance

A security extension to the GridStat managed pub-sub system has been developed based on the security architecture described above. Several proof-of-concept modules were developed, many based on Sun's *Java Cryptographic Extension* (JCE), see Table 1. JCE has a very limited interface that is not compatible with GridStat's event publication format. As a result the modules employ costly translation logic between the two formats that copies the data at least two times. This means that the numbers presented in Tables 1 and 2 have a lot of potential for improvement. The latency requirements for electric power substation automation, range from 4ms to 16ms internal to

a substation and from 8ms to 10s external to a substation [8]. So, even with room for improvement the security architecture is able to provide security with low added latency.

By employing an end-to-end security approach the latency associated with each module is independent of the length of the paths that events travel so the added latency is presented without taking the path length into account in Table 1. Since the size of the event obviously affects the number of processor cycles each module needs to perform its task, an experiment is run for each of the modules with messages containing a single integer update and messages containing a 200 byte string update. The 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, or small collections of primary types, but in rare cases larger events would be needed and thus they are supported.

The data plane performance experiments were run on a simple GridStat topology having a single leaf-QoS Broker, one leaf-SMS, one status router, one publisher, one subscriber. 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 Linux with kernel version 2.6.20-16. All GridStat components were 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**
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
DES[1]	8,059	22,945	9,561	42,486	17,620	65,431
TripleDES[1]	9,748	43,642	12,473	70,439	22,221	114,081
OneTimePadObf	8,303	34,314	7,489	28,314	15,792	47,959
AESObfusccation[1]	52,804	57,260	34,134	58,800	86,938	116,060
SimpleAuth	2,161	2,189	2,332	2,414	4,492	4,603
RSA[1]	$114*10^6$	$114*10^6$	933,402	944,225	$114*10^6$	$114*10^6$
Crc	2,589	16,789	2,654	17,075	5,243	33,863
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` [18] 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, added an average of 65 microseconds latency on the 200 byte event stream and `TripleDES` used as much as 114 microseconds. Clearly the `AES` and `Blowfish` modules are the better choice for achieving confidentiality.

The two evaluated obfuscation modules follow two different approaches to obfuscation. The `AESObfuscation` module randomly generates a new key for each event and uses this key to encrypt the event 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 carries a great performance cost. Since

[1]Modules implemented based on Sun's JCE [20, 3]

a new key is used for each status update, the module needs to re-initialize its encryption cipher for each event, and the performance is much worse than that of the `AES` encryption module that uses the same key each time. This corresponds to the findings of Opyrchal and Prakash [14]. The `OneTimePadObf` module, on the other hand, performs much better, especially with small events. It generates a one-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 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 far exceeds the latency requirements for some power system applications, while the `SimpleAuth` module, because of its simplistic logic only adds 4.5 microseconds in the 200 bytes experiment.The `SimpleAuth` algorithm simply adds 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, the experiment based on it serves as a performance comparison measure.

The four integrity modules implemented and evaluated are error checking modules that use different hash algorithms to assert the integrity of received information. The `Crc` module uses a 16 bit *cyclic redundancy check* (CRC) optimized for small events, while the `MD5ErrorCheck` [17] module generates a 128 bit hash, the `SHAErrorCheck` [7] uses 160 bit hash, the `SHA512ErrorCheck` module employs a 512 bit hash 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 200 byte events on average can be error checked with either a MD5 hash or a SHA hash within 20 micro seconds of added end-to-end latency.

To test how the performance was affected by combining modules into sets the experiments presented in Table 2 were undertaken. As the two first experiments show, the order in which modules are used affects the latency that 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.

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 integer events with an average added latency of below 32 microseconds (120 microseconds for the 200-byte event stream). Experiment 5, on the other hand, shows that if we are interested group membership authentication instead of error checking then the added latency can be reduced to 85 microseconds for the 200 bytes stream.

To show a worst case scenario experiment 6 combines the three slowest 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 microseconds for 200 bytes. This clearly is not an optimum combination of modules, but nevertheless provides results that are tolerable in a wide area setting.

#	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	60,111	31,322	120,813
5	SimpleAuth OneTimePadObf Blowfish	13,393	46,717	11,392	39,115	24,785	85,833
6	AESObfuscation TripleDES SHA512	70,602	135,048	59,304	108,073	129,906	243,121
8	SimpleAuth OneTimePadObf Crc Blowfish	15,765	121,353	13,280	96,782	28,348	218,136
7	SimpleAuth OneTimePadObf Blowfish SHA	15,068	59,583	16,577	49,986	32,342	109,569

Table 2. Module set latencies in nanoseconds

Experiment 7 takes it one step further and shows that we are able to provide authentication, obfuscation, encryption and integrity while adding only 32 microseconds of latency for the integer stream and 110 for the 200 bytes stream by combining the `SimpleAuth`, `OneTimePadObf`, `Blowfish` and `SHAErrorCheck` modules. This clearly shows, even with these non-optimized, proof-of-concept modules, that the modular approach is able to provide the security needed for the event streams while adhering to real-time industry standards in critical infrastructure such as the IEC 61850 4 millisecond inter-intelligent controller requirement [9]. Future optimization will only further reduce the added latency and enable even higher levels of encryption.

5 Related Work

Publish-subscribe research tends to focus on performance, scalability and expressiveness [19] and the research on security related matters that is being done is mainly concentrated on *content-based* publish subscribe (CBPS)[13, 14, 15, 16]. In CBPS systems the subscribers register filter functions and events are routed based on evaluations of these functions applied to 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 challenging research questions. Most published approaches to these requirements have been variations on encrypting each attribute of an event with a different key, thus allowing brokers with different needs to decrypt just what is needed to route the event successfully [14]. Khurana [13] 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 [16].

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. [19] 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. Pesonen et al. [15] present a security architecture for multi-domain CBPSs that acknowledges the fact that providing confidentiality in multiple security domains warrants special considerations, such as 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.

A common trait of the security architectures published for CBPS is that they all employ PKI in some fashion. This introduces weaknesses that were explored in Section 3.2.1.

6 Conclusions

This paper has presented a security architecture that, through the use of transparent interchangeable software modules, can provide confidentiality, integrity, obfuscation, authentication and filtering to managed pub-sub systems. This is accomplished while adhering to critical infrastructures' real-time requirements by taking advantage of managed pub-sub systems ability to support an end-to-end approach that is not available in CBPSs.

The architecture's support for secure re-keying and re-moduling of both data plane and management plane communication, without relying on root-certificates, provides service integrity without locking the system to a single set of security algorithms. The leaf-SMSs provide publication con-

fidentiality by only allowing authenticated subscribers with the right credentials to access the needed modules and keys to read the event streams.

It is not possible to predict the advances in technology and hence the types of attacks possible in future. However, having an architecture independent of specific modules and keys allows evolution of the system to adapt to new techniques and to resist even currently unknown attacks.

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. Together this makes the use of the presented security architecture a ideal tool for providing managed pub-sub with the security it needs to be utilized in critical infrastructures.

7 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 National Science Foundation Grant CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)).

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, URL: http://www.microsoft.com/technet/prodtechnol/ windows2000serv/reskit/dist

[3] Sun Microsystems. Java SE Security, Sun Developer Network URL: http://java.sun.com/javase/technologies/security Last checked 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. Gjermundrød, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical Report EECS-GS-009, Washington State University, May 2007.

[9] F. Cleveland. IEC TC57 security standards for the power system's information-infrastructure beyond simple encryption. URL: http://xanthus-consulting.com/Publications/, June 2007.

[10] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[11] K. H. Gjermundrød. *Flexible QoS-Managed Status Dissemination Middleware Framwork for the Power Grid*. PhD thesis, Washington State University, August 2006.

[12] 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.

[13] H. Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In *SAC*.

[14] 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.

[15] 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.

[16] 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.

[17] R. Rivest. RFC 1321: The MD5 message-digest algorithm, April 1992. MIT Laboratory for Computer Science and RSA Data Security, Inc.

[18] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204. Springer-Verlag, December 1993.

[19] 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.

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