Trusted Group Key Management For Real-Time Critical Infrastructure Protection

Jonathan Jenkins, Sean Easton, David Guidry, Mike Burmester, Xiuwen Liu
Xin Yuan, Joshua Lawrence, and, Sereyvathana Ty*

Department of Computer Science, Florida State University, Tallahassee, Florida, 32306, U.S.A.
{jenkins, easton, dguidry, burmester, liux, lawrence, xyuan}@cs.fsu.edu

*Sandia National Laboratories, New Mexico,Albuquerque, NM 87185, U.S.A.
sty@sandia.gov

***Abstract*—Most critical infrastructures can be modeled as cyber-physical systems whose cyber components control physical processes so as to optimize specific system objectives. Protecting such systems from malicious threats (including insider threats) is particularly challenging. One solution, based on Trusted Computing technologies such as the Trusted Platform Module (TPM), uses an infrastructure that ensures that only trusted programs are executed. Such technologies readily support secure unicast communication. However, many critical infrastructures employ multicast. Addressing multicast requires attention to (a) compatibility and (b) real-time compliance. In particular, sealed multicast storage for which access takes no longer than unicast. We present a trusted computing architecture for multicast communication based on an adaptation of the Kerberos authentication service along with TPM trust engines. This architecture is efficient and scalable (one session key per multicast channel). We show that, by integrating our framework with an open source IEC 61850-90-5 profile emulator for power utility automation, synchrophasor data feeds are protected in real-time ($<$ 4ms, the IEC61850-90-5 threshold) against strong adversaries.**

***Index Terms*—Group key management, trusted multicast, Kerberos, trusted platform modules, IEC 61850-90-5.**

I. Introduction

With recent improvements in model-based reliable software development and the increased efficiency of computing devices, critical infrastructures that utilize computer algorithms to optimize various aspects of physical processes are emerging from specialized domains to much broader areas with many significant applications. These systems exploit the flexibility (*i.e.,* programmability) and the distinctive operational characteristics (*e.g.,* the number of computer instructions per unit energy consumed) of cyber components, and are becoming a key factor in improving sustainability of human activities.

Critical infrastructures must be designed to remain stable even when some of their components fail because of accidental or malicious actions. For critical infrastructures, protection must extend to nation state actors and insiders. To secure a critical infrastructure it is essential that its communication channels are protected in real-time: correct messages delivered at the wrong time could lead to erroneous system responses and even failures. This renders most security paradigms for cyber-only systems inapplicable. For example, confidentiality, integrity and availability are the three primary goals of cyber security systems with confidentiality being most important. However, due to the robustness and resilience requirements for cyber-physical-systems, availability and integrity are the primary goals with confidentiality often only a secondary goal.

Group key management protocols for cyber systems have been widely studied due to their important role in achieving confidentiality, integrity, and access control; see [9], [4] for recent surveys. There is little work on group key management for cyber-physical systems. Chen et al. [2] proposed a wireless key management scheme to secure the Internet of Things but did not consider real-time requirements of cyber-physical systems. Barker et al. [1] proposed a framework for designing key management systems, but special requirements of cyber-physical systems are not fully considered. A profile emulator for IEC 61850-90-5-compliant systems [10] was recently made available with an intended group key management protocol. However, the group key management has not been implemented. Similarly, there is little work on integrating Kerberos and TPM architectures to enhance security with the exception of [5], where Leicher et al. use a TPM to enhance Kerberos exchanges by building tickets with data linked to a single TPM, and checking integrity information before granting tickets. Our work differs from [5] in several aspects, most notably in that Leicher et al. do not handle group keys.

In this paper, we propose a novel trusted real-time group key management framework for critical infrastructures. We leverage the widely deployed Kerberos authentication service and the Trusted Platform Module (TPM) interface as building blocks. In particular, we utilize the Key Distribution Center (KDC) of Kerberos to authenticate the principals (users or components) of a cyber-physical system, which allows us to establish authentication channels between the KDC and any principal. On this foundation, we extend the KDC with the ability to create multicast channels for efficient group session key distribution, supported by TPM storage of long-term keys.

To demonstrate that the proposed group key management protocol is robust in real-time we analyze its performance and sufficiency by using workstations that emulate IEC 61850-90-5-compliant protocols for substation automation systems.

The rest of the paper is organized as follows. In Section II we provide background for Kerberos and the TPM interface, and a brief survey of related work. In Section III, we describe the proposed real-time group key management framework. In

Section IV we illustrate the effectiveness of the framework by employing it to secure an emulation of a IEC 61850-90-5-compliant power automation system. We conclude in Section V.

II. Background

Our framework is built on the Trusted Computing (TC) paradigm (in particular the TPM interface) that provides built-in and non-migratable trust, and the Kerberos network authentication service. In this Section we briefly summarize the TC and Kerberos architectures, and discuss related work on trusted key management for cyber-physical systems.

A. The Trusted Platform Module

The Trusted Computing Group (TCG) [11] describes an architecture in which *trusted engines*, called *roots of trust* (RoT), are used to establish trust in the expected behavior of the system. The Trusted Platform Module (TPM) is an interface that uses trusted engines for binding data to platform configurations of hardware systems to enhance software security.

A TPM [13] is implemented as a hardware or software cryptographic device with several critical capabilities: remote attestation, secure storage, and integrity measurement. It provides a range of cryptographic primitives including: a random number generator; hashing functions; asymmetric encryption/decryption; two unique asymmetric non-migratable key pairs (set at the time of manufacture): an attestation identity key pair for signing data originating at the TPM, and an endorsement key pair for decrypting owner authorization data and messages associated with the attestation key creation; symmetric keys to bind small amounts of data (typically keys) and to authenticate transport sessions. The TPM also has a small amount of storage (mainly for keys). There are three mandatory RoT in a TPM: a RoT for integrity measurements (RTM), a RoT for storage (RTS) to protect keys and entrusted data, and a RoT for reporting (RTR) to (a) expose shielded integrity measurements and (b) attest to the authenticity of stored values. Security is based on an integrity protected boot process in which executable code and associated configuration data is measured *before* it is executed—a hash of the BIOS code is stored in a Platform Configuration Register (PCR).

For remote attestation the TPM uses the attestation key to assert the state of its current software environment to a third party (by signing current PCR values). For sealed storage, encryption/decryption/authentication keys are released from protected storage, conditional on the current software state (using current PCR values). In our group key management framework we utilize the RTS capability of the TPM (key *sealing*) to enhance the security of long-term keys.

B. Kerberos

Kerberos 5 [8] is a single-sign-on authentication service for open (untrusted) networks based on the Needham-Schroeder authentication protocol [7]. It defines a highly dynamic trust graph with short lived authentication edges. With Kerberos, any client principal (a workstation user or network Server) can obtain credentials from a trusted Key Distribution Center (KDC) for authenticated access to an application Server. The KDC has two distinct functionalities: authentication and key distribution. These are realized by an Authentication Server (AS) and a Ticket Granting Server (TGS), which typically are implemented as different protocol entry points on a Kerberos Server [6].[1]

We use the RFC 4120 [8] specifications to describe the Kerberos authentication service, and in particular the data structures and message formats, which we extend in Section III to capture multicast communication.

There are two formats for Kerberos exchange messages:

1) KRB_KDC_REQ $<$*cname, sname, padata, ctime, nonce, kdcoptions*, $\ldots>$:[2] a request format with the client's name in *cname*, the Server's name in *sname*, pre-authentication data in *padata*, the client's time *ctime*, a nonce (to prevent replay attacks) and the KDC options *kdcoptions* $<$ *msgtype, encauthdata, from, till,*, $\ldots >$ with *msgtype* either KRB_AS_REQ or KRB_TGS_REQ, *encauthdata* used only in KRB_TGS_REQ, *from* the desired start time, *till* the requested expiration time, $\ldots$.
2) KRB_KDC_REP $<$ *msgtype, padata, cname, encdata*, $ticket, \ldots>$: a reply format where *msgtype*, *padata*, and *cname* are the same as in the request; $encdata <$ $key, sname,$ $lifetime, \ldots>$ that contains an application secret session key encrypted with the client's secret key; and $ticket < sname, encpart >$ that contains the encrypted part $encpart <key, cname, sname, time, lifetime, \ldots>$ with a copy of the session key and the local KDC time encrypted with the key of the end Server.

When a client requires authenticated access to an application Server, it first gets authenticated by the AS. For this purpose it sends a request KRB_AS_REQ to the AS (in the KRB_KDC_REQ format) with its name, local time, requested lifetime and a nonce in the clear (not encrypted). Upon receiving the request, the AS verifies that the client has a record in the Kerberos database. If there is a record (that contains the client's private key) and if the client's timestamp is close to the local time of the AS then the AS generates a random key, the *initial session key*, and sends to the client principal either:

- a KRB_AS_REP reply message with the initial session key in *encdata*, and a Ticket Granting Ticket (TGT) in *ticket* that contains a copy of the initial session key encrypted with the secret key of the TGS; or
- a KRB_ERROR $<$*cname, stime, edata* $>$ with a request for pre-authentication by the client.

In the second case, the client must resend a request message KRB_AS_REQ, but this time with an authenticator in *padata*. The authenticator is an encryption of the client's timestamp

[1] These must be secured independently: a separation-of-duties security policy to mitigate insider threats.

[2] We only list those fields that are used in our approach. The full list is available from [8].

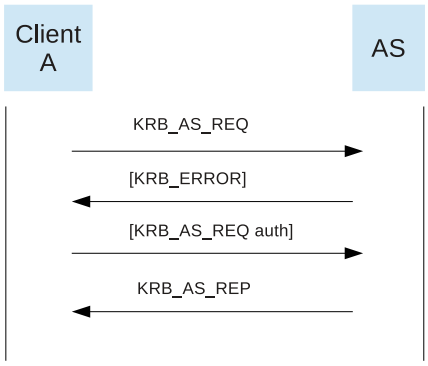


Fig. 1. Authentication Service Exchanges.

with the client's key. We illustrate in Figure 1 the messages exchanged by the client and the AS.

The first round of the Kerberos protocol establishes an authentication channel between the client and the TGS.

To get authenticated access to a Server the client sends a request message KRB_TGS_REQ to the TGS with a valid TGT in *encauthdata*. Upon receiving a request the TGS decrypts the TGT to check that it is valid (and not expired) and get the initial session key. If the TGT is valid then the TGS generates a new random key, called the *service session key*, and formulates a reply KRB_TGS_REP that contains the service session key, the Server's name and the lifetime of the ticket in *encdata*, encrypted with the initial session key, together with a service ticket that contains a copy of the secret session key, the client's and Server's name, the ticket lifetime and a KDC timestamp, encrypted with the secret key of the Server. This second round of the Kerberos protocol establishes an authentication channel between the client and the application Server.

There are several security threats that Kerberos protects against. Replay attacks and man-in-the-middle attacks are addressed by using nonces, timestamps and authenticators. The privacy of the initial session key and the service session key are assured by using private channels (realized by encrypting with the client's private key and the AS session key, respectively). However, Kerberos does not protect against denial of service attacks on the KDC caused by flooding it with authenticated requests. This will slow down the response time to legitimate clients. Furthermore Kerberos does not address insider threats, or bad implementations.

A TGT is typically valid for 8-24 hours, allowing the client to get several service tickets from the TGS for authenticated access to network services during this period—single-sign-on.

III. Multicast Group Authentication

Kerberos is a single-sign-on authentication service for client-server applications. It is not designed for multicast applications where a client wants authenticated multicast access to several principals, a common requirement with cyber-physical systems. Furthermore it does not address denial of service threats or insider attacks, which can incapacitate power grids and more generally critical infrastructures. In this section we show how to extend Kerberos to capture both requirements: efficient authenticated group multicast as well as security against denial of service threats and insider attacks. Our extension is described in terms of the RFC 4120 [RFC 4120] specification. For security we use the TPM interface (Section II-A).

A. Protocol Description

The protocol consists of several sub-protocols that address particular cases. In Section III-B we consider the case when the group of principals is listed in the Kerberos database, and in Section III-C when the group of principals is not listed. In Section III-D we show how to secure a Kerberos multicast group authentication protocol and in Section III-E we address reliability and denial of service (DoS) issues.

Suppose a client requires authenticated multicast access to a group GRP of principals and that the client has a valid TGT (obtained as in Section II-B). We distinguish two cases:

1) GRP and its long term secret group key is listed in the Kerberos database;
2) GRP and its long term secret group key is not listed.

B. GRP is listed in the Kerberos database

In this case the client gets a session group key for GRP by sending a request message KRB_TGS_REQ to the TGS with the name GRP in *sname*, a valid TGT in *encauthdata*, and if necessary an authenticator in *padata*.

Upon receiving the request the TGS decrypts the TGT to verify that it is valid and not expired, that the timestamp is close to the local Kerberos time, that the group of principals in GRP is listed in the Kerberos database (with a long term group key), and get the initial session key. If the TGT is valid then the TGS generates a random key, the *group session key*, and formulates a reply message KRB_TGS_REP that contains the group session key and the name of the group in *encdata* (encrypted with the initial session key), and a service group ticket that contains a copy of the group session key in *ticket* (encrypted with the long term group key of GRP). The TGS then sends the reply KRB_TGS_REP to the client principal.

The client can now get authenticated multicast access to the principals of GRP. For this purpose two Kerberos client/application message authentication types are used:

1) KRB_AP_REQ $< cname, sname, ctime, ticket, \ldots >$,
2) KRB_AP_REP $< cname, sname, ctime, encpart, \ldots >$.

The client formulates a request GRP_AP_REQ with the name GRP in *sname* and the group ticket obtained from TGS in *ticket* (encrypted with the long term secret group key of GRP), and multicasts the request to group GRP.

Upon receiving the request, each one of the principals in GRP after checking the group name in *sname* decrypts the group ticket to get the session group key, and then formulates a reply message KRB_AP_REP with GRP in *cname* and, its name and a timestamp in *encpart* (encrypted with the session group key).[3] This serves as a confirmation. The GRP principal

[3]A TPM-compliant principal will not impersonate other principals, as we shall see in Section III-D.

then sends the reply message KRB_AP_REP to the clier principal.

C. GRP is not listed in the Kerberos database

In the second case the TGS must establish a long ter group key and distribute this key to all principals of GR This requires the TGS to establish an authentication channe with each one of the principals in GRP and use this t transport a long term group key. As in Section III-B we sha use the two message authentication types KRB_AP_REQ an KRB_AP_REP for this purpose. Only in this case the clier is the TGS.

To establish a long term group key the client principal first formulates a request KRB_TGS_REQ that contains the name GRP and a list of all the names of its principals in *sname*, and sends the request to the TGS. Upon receiving the request, the TGS first verifies that this is valid. If so, and if the group GRP is not listed in the Kerberos database, then the TGS selects a random long term group key and sends to each one of the principals of GRP an application request message KRB_AP_REQ with:

1) the name GRP and the names of its principals (as supplied by the client) in *sname*;
2) the long term group key in *ticket*, encrypted with the secret key of the principal.

Each principal in GRP, after checking KRB_AP_REQ for correctness, and that its name is included in the list of principals, decrypts the *ticket* to get the long term group key, and then formulates a reply KRB_AP_REP with the name GRP in *cname* and the principal's name and a timestamp in *encpart* (encrypted with the secret key it shares with the TGS). This serves as a confirmation. The principal then sends the reply to the TGS.

Multicast authentication is achieved as in Section III-B. The TGS selects a random group session key and sends to the client a reply message KRB_TGS_REP in which the name GRP and the group session key are in *encdata* and the group session key is in *ticket*, encrypted with the long term group key of GRP. In this way the client shares with the principals of GRP the session group key. We illustrate the message exchanges between the client, the TGS and the principals of GRP in Figure 2.

Policies regarding allowable groups of principals must be adhered to. Since the number of possible groups can be very large, one may wish to have a separate Server, the group Ticket Granting Server manage group authentication services. We shall further discuss this issue in Section III-E

D. Trusted Group Authentication

To secure the Kerberos multicast authentication protocol we use a TPM architecture. We assume that the principals of the Kerberos system and the KDC Servers are TPM-compliant. This guarantees that the private keys of the principals in the Kerberos database are in shielded locations and only become available when needed by Kerberos. Furthermore, prior to establishing an authentication link, remote attestation will

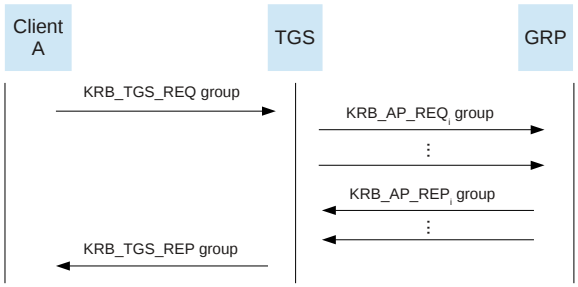


Fig. 2. Multicast Group Authentication

ensure that there are trust paths that link the KDC to the client and the principals (via shared secret keys). This will protect the system from external integrity and confidentiality threats, as well as insider threats that target the private long term keys of principals.

The Trusted Network Connect (TNC) [12] is a TC platform interoperability architecture for trusted access control that is based on the TPM. It allows for network access control based on reliable host integrity information (*e.g.*, from the TPM). For our Kerberos application this means that, an authentication channel between a client and Server is established only if:

1) The identities of the client and Server are trusted. For the first and second rounds of the Kerberos protocol, the Server is a trusted entity (either AS or the TGS). If the client is pre-authenticated, then its identity is confirmed via a secret shared key.
2) The client is allowed access to the Server. Authorization can be implemented either with Kerberos 5, or by using an appropriate Access Control System with credentials.
3) The identity of the client and Server is authenticated. This is achieved by encrypting data with a shared secret key (either the secret key of the client, or a key generated by the KDC). This requires a RoT engine to be invoked on the TPM of both sides to release the required encryption key. The TPM will only release keys if the current configuration of the OS of both principals allows for this.

The Kerberos 5 specifications [8] also allow for a public key setting. For such settings public key authentication channels are used to establishing a symmetric key authorization service (in this case the KDC only knows the certified public keys of the clients). One of the advantages of such an approach is that Kerberos is not required to store long term secret information.

E. Reliable Group Authentication

As the number of Kerberos principals grows, scalability becomes an issue. For a reliable service the KDC must be able to handle a large number of requests in real-time. One way to address this issue is to have a realm with multiple KDCs and load balancers to distribute stressed traffic across the KDCs. This approach tends to maximize network throughput

while minimizing response time. However, we now have a synchronization problem: KDC databases must be synchronized in real-time. For example, if a principal is blacklisted on one KDC, then all the other KDCs should also blacklist this particular principal for the same time period.

There are several ways to address synchronization. One way is use a Lightweight Directory Access Protocol (LDAP) [RFC4511]. This supports multiple masters in replication mode, and allows data to be stored by a group of network entities, and updated by any network entity. By combining the Kerberos protocol with LDAP, we allow any KDC to update its database, with the updated data being propagated to all other KDC databases. Since LDAP supports Secure Socket Layer (SSL), we can easily transfer the updated data through a secure socket.

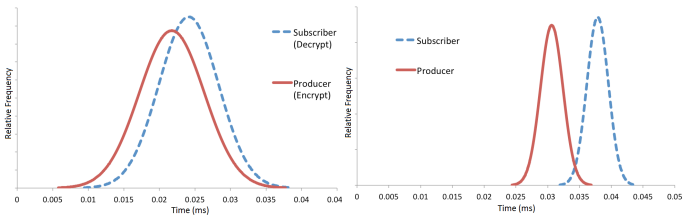
If we assume that a TNC architecture with multiple KDCs is used for our Kerberos implementation then we can achieve a satisfactory degree of reliability for most applications. However, for critical infrastructure protection, such as the electrical grid, Quality of Service (QoS) may not provide adequate security in real-time, particularly when the system has to respond to critical events in milliseconds. For such cases we need guaranteed availability in real-time: for the electrical grid this is typically $4\,ms$ (rate $r = 250$ packets/second).

Our approach to secure the Kerberos multicast framework is based on the following requirements:

1) *Reliability*. The Kerberos network system has sufficient redundancy so that the probability of system failure caused by Nature is less than a small threshold ε.[4] This requires that we: (i) specify the set of natural fault events $\mathcal{F}$, (ii) estimate the probability ε that such events occur, and (iii) replicate system services sufficiently (and independently) to enable it to respond with probability $\geq 1 - \varepsilon$ to at least r requests per unit time.
2) *Integrity and confidentiality.* The Kerberos system is TPM-compliant. This prevents integrity and confidentiality attacks.
3) *Availability.* We assume that the supported rate r is sufficient to cope with high priority requests. This requires a bound on the number of high priority requests, particularly on (i) the number of requests per KDC Server and (ii) the number of clients per Server. To protect against DoS attacks caused by flooding bogus requests, an intranet is used and the high priority designation of a request is authenticated (*e.g.,* included in *padata*)—in particular, only high priority requests get forwarded when the traffic level is high.

The level of security provided by adopting these requirements is that afforded by the TPM interface (in particular how well it is implemented), provided that faults caused by Nature are restricted to the event space $\mathcal{F}$.

[4]For critical infrastructure protection, typically $\varepsilon = 2^{-30}$.



(a) (b)

Fig. 3. (a) AES Encryption & Decryption time, (b) HMAC Generation & Authentication time on the producer & subscriber.

IV. Trusted Substation Automation

To validate our group key management protocol and establish real-time efficiency, we integrated it with an open source IEC 61850-90-5 emulation program recently released by SISCO [10] for trusted power system automation applications.

A. System Setup

Our experimental software is based on the open source profiler for the IEC 61850-90-5 Technical Report released by SISCO [10], which we enhanced using the techniques in Section 3. The modifications to the source code include porting the code to Linux, adding our group key management protocol, a trusted platform module, cryptographic support for encryption and authentication, and a time-keeping functionality for data generation. In order to read time accurately we used a function written in assembly to access the CPU tick counter directly on one machine. For synchronization, packets that relate local CPU ticks were exchanged by the machines and a best fit obtained by using a least square algorithm.

A testbed consisting of seventeen local machines connected by a Cisco Catalyst 3560G series PoE 48 switch was used. All machines were running Ubuntu Linux 12.04 64 bit OS; eight had an Intel Xeon 5120 1.86 GHz x2 CPU with 2GB of memory and nine had an Intel Xeon E5506 2.13GHz x4 CPU and 6 GB of memory. A dedicated machine running the krb5 API acted as our Kerberos 5 release 1.10 server, which interfaced directly with the Kerberos API.

B. Building Blocks

Encryption/Decryption. The producer and subscribers perform an AES encryption or decryption on the packet's payload. As shown in Figure 3(a), over the course of 500 (GOOSE) packet transmissions the average time for encryption was $0.02175\,ms$ with a standard deviation of $\sigma = 0.00457\,ms$. Decryption took on average $0.02413\,ms$ with $\sigma = 0.00420\,ms$.

HMAC Authentication. The producer and subscribers calculate an HMAC using SHA2. As shown in Figure 3(b), the average time required over 500 packets by the producer was $0.03063\,ms$ with $\sigma = 0.00178\,ms$. The subscriber's average time was $0.03786\,ms$ with $\sigma = 0.00167\,ms$.

RNG. The average time to generate a random number was $0.0108715\,ms$ with $\sigma = 0.000399\,ms$.

TPM lock/unlock. The average time to lock a TPM key was $0.48788\,ms$ with $\sigma = 0.021401\,ms$, and to unlock a TPM key $0.36454\,ms$ with $\sigma = 0.051601\,ms$.

C. Initial Group Registration

We estimated the time required to perform the initial key registration as described in Section III-C. If rng is the time required to generate a random number, enc, dec the time to encrypt and decrypt the payload of a packet respectively, mac the time to authenticate a packet, $unlock$ the time to unlock a TPM key, and n the number of principals in a group, then the time required by TGS to distribute a long term group key is given by $f(n) = rng + enc \cdot (n+1) + dec + unlock/lock$, since the producer must select a random number, encrypt it with n different keys, and then each one of the subscribers in the group must unlock their secret key, decrypt the received packet, and send a confirmation. Both the producer and the subscribers must then relock their secret keys, and the distributed long term group key. Using our earlier average time estimates we get: $f(n) = 0.035 + 0.02175\,(n+1) + 0.365\ (\text{unlock}/\text{lock})\ ms.$

D. Session Group Key Distribution

The time required to update a session key, as described in Section III-B, is: $rng + 2 \cdot enc + dec + unlock\ ms$, since the producer must select a random number, encrypt it twice (for the subscriber and the TGT), and the subscriber must unlock a TPM secret key and then decrypt the packet. Again both the producer and the subscribers must relock their secret keys, and the distributed long term group key. Using our earlier estimates the average time is: $0.0785 + 0.365\,(\text{unlock})\ ms.$

E. Transmission of GOOSE Packets

The producer must build the packet, encrypt the payload (for confidentiality), calculate an HMAC (for authentication) and finally transmit the packet. The producer must build the packet, encrypt the payload (for confidentiality), calculate an HMAC (for authentication) and finally transmit the packet. Each subscriber that receives the packet must verify the HMAC and decrypt the payload. The time required for this is: $enc + dec + 2\,mac$ since the producer must encrypt the payload, authenticate it, and the subscriber must verify it and then decrypt it.

Using our earlier average time estimates we get that the processing time for a GOOSE packet is: $0.02175 + 0.02413 + 0.03063 + 0.03786 = 0.11437\ ms$. This does not include the time for network transmission, the IEC 61850-90-5 header generation, the IP protocol formatting, and other general processing requirements. We estimated this to be roughly an additional $0.00069\,ms$ for the producer and $0.54\,ms$ for the subscriber. The relatively high value for the subscriber is due to queuing delays (which also impacts on the standard deviation $\sigma = 0.302\,ms$).

The total processing time is still no more than $1\,ms$, which is well within the $4\,ms$ time limit for GOOSE packets of the IEC 61850-90-5 Technical Report.

V. Conclusion

We have proposed and developed a framework that enables secure multicast communication via efficient and scalable group key management. Leveraging the widely used Kerberos authentication service and TPM modules for secure storage, our system provides distinctive security advantages. We demonstrate the sufficiency of the proposed framework by *implementing* a fully integrated and functioning system based on a power utility automation profile emulator. Our emulation results show that the system can meet the stringent real-time requirements of the IEC 61850-90-5 Technical Report while providing secure guarantees.

The importance of security for critical infrastructures and cyber-physical systems is widely recognized and our framework and implementation are a first step toward securing cyber-physical systems. While the current experiments involve a relatively small number of nodes (due to practical limitations), the results can be readily generalized to much larger systems. For example, there are an estimated 5,000 substations in the Eastern Interconnection with over 40 intranets in the United States. Assuming that each intranet is organized as a network without any hierarchy, the inherent scalability and efficiency of multicast means the performance of sending to 125 nodes should be comparable to a few nodes. When intranets are organized into a hierarchy, additional delays will be limited to a few hops. Our results therefore provide meaningful measurements for the entire grid, not just small network settings.

Acknowledgement The paper is based upon work supported by the National Science Foundation Grant No. 1027217.

References

[1] E. Barker, D. Branstad, S. Chokhani, and M. Smid. A Framework for Designing Cryptographic Key Management Systems. *NIST DRAFT Special Publication 800-130*, 2010.

[2] Dong Chen, Guiran Chang, Dawei Sun, Jie Jia, and Xingwei Wang. Lightweight key management scheme to enhance the security of internet of things. *Int. J. Wireless and Mobile Comp.*, 5(2):191–198, 2012.

[3] International Electrotechnical Commission. IEC/TR 61850-90-5, Edition 1.0 2012-05, Technical Report, Power systems management and associated information exchange Data and communications security. *http://webstore.iec.ch/preview/info_iec61850-90-5%7Bed1.0%7Den.pdf*, May 2012.

[4] Bibo Jiang. A survey of group key management. *Int. Conf. Computer Science and Software Engineering*, 994–1002, 2008.

[5] Andreas Leicher, Nicolai Kuntze, and Andreas U. Schmidt. Implementation of a trusted ticket system. *IFIP Advances in Information and Communication Technology*, 297:152–163, 2009.

[6] D. Moberg and R. Drummond. RFC 4130, MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2, 2005.

[7] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[8] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC 4120, The Kerberos Network Authentication Service (V5), 2005.

[9] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Comp. Surveys*, 35(3):309–329, 2003.

[10] SISCO. Cisco and SISCO Collaborate on Open Source Synchrophasor Framework, Press Release. *http://www.sisconet.com/downloads/90-5_Cisco_SISCO.pdf*, 2011.

[11] Trusted Computing Group. http://www.trustedcomputinggroup.org/.
[12] Trusted Network Connect Architecture for Interoperability (TNC), Specification 1.3. Revision 6, April 2008.
[13] Trusted Platform Module (TPM) Structures, Level 2, Version 1.2. Revision 116, Communication Networks and Systems for Power Utility Automation. *http://www.trustedcomputinggroup.org/resources /tpm_main_specification*, March 2011.