IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 18, 2019, accepted February 26, 2019, date of publication March 14, 2019, date of current version March 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2902571

Performance Evaluation and Analysis of IEC 62351-6 Probabilistic Signature Scheme for Securing GOOSE Messages

SHAIK MULLAPATHI FAROOQ[1], (Student Member, IEEE), S. M. SUHAIL HUSSAIN[2], (Member, IEEE), AND TAHA SELIM USTUN[2], (Member, IEEE)

[1]Department of Computer Science and Engineering, YSR Engineering College, Yogi Vemana University, Kadapa 516360, India
[2]Fukushima Renewable Energy Institute, AIST (FREA), Koriyama 963-0298, Japan

Corresponding author: Shaik Mullapathi Farooq (smfarooq@ieee.org)

This work was supported in part by the Research and Innovation Fund 2018 and in part by the KEIDANREN (Japan Business Federation) Promotion of Environmental Protection Foundation's Research Grant-2018.

ABSTRACT Cyber security is a growing concern in power systems. To achieve security requirements such as authentication and integrity for generic object-oriented substation event (GOOSE) messages, IEC 62351-6 standard recommends using digital signatures. Furthermore, it explicitly specifies to use RSASSA-Probabilistic Signature Scheme (PSS) digital signature algorithm based on RFC 3447. Power systems run in real-time and implemented cybersecurity measures have to strictly meet timing requirements. Therefore, it is very important to study performances of such methods and contrast them with the timing requirements stipulated by grid operations, e.g., power system protection enforces a maximum delay of 3 ms. In this fashion, it can be analyzed whether a recommended cyber security mechanism is fit for use in power systems. In previous works, only RSA digital signatures were studied and its performance evaluation in terms of computational times for securing GOOSE messages have been studied. This paper analyses the timing performance of RSASSA-PSS digital signature algorithm for securing the GOOSE messages. This is important to assess its feasibility for IEC 61850-based networks, as specified by the IEC 62351-6 standard. RSASSA-PSS digital signature algorithm is implemented in Python and verification times are calculated. The results show that RSASSA-PKCS1-v1_5 1024 key digital signatures provide improved performance compared with other RSA digital signature schemes. That being said, none of the algorithms is fast enough to be implemented for time-critical operations such as protection coordination.

INDEX TERMS Cyber Security in power systems, probabilistic signature scheme (PSS), generic object-oriented substation event (GOOSE), public key cryptographic standard1 version 1.5 (PKCS1-v1_5).

I. INTRODUCTION

Taking advantage of latest advancements in information and communication technologies, the legacy power system is transforming as smart grid. The smart grids are more resilient, self-healing, environmentally friendly, efficient, provides higher power quality and more options for customers. Substations play a very important role in building of smart grid. To achieve the goals of smart grid a reliable, secure and interoperable communication for substation is required. The existing supervisory control and data acquisition (SCADA) and remote terminal unit (RTU) systems deployed in substation are not capable to address these concerns [1].

IEC 61850 has emerged as a de facto standard for substation automation [2]. Due its object-oriented and interoperable design, it is very versatile in modeling different power system equipment. IEC 61850 specifies different protocols such as Generic Object-Oriented Substation Event (GOOSE), Sample Value (SV) and Manufacturing Message Service (MMS) for exchanging information related to different services [3]. Recently, cybersecurity concerns regarding IEC 61850 based substation automation systems have been largely reported in literature for substations [4]–[7] as well as phasor measurement units [8]. Of the different IEC 61850 messages, the

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandra De Benedictis.

security of GOOSE messages is more critical as it carries time-critical information related to power system operation. If the security of GOOSE message is compromised, it may have catastrophic effects on power system operation [5].

Security requirements for IEC 61850 GOOSE messages are addressed by IEC 62351-6 standard [9]. It is clearly stated that authentication and integrity of GOOSE messages are important security requirements. The solution specified by IEC 62351-6 is use of digital signatures that are signed by RSA algorithms. According to this standard, the variant of RSA used for this purpose has to strictly follow RFC 3447 [10] and be compatible with RFC 2313 [11].

Furthermore, IEC 62351-6 specifies that the confidentiality of GOOSE messages cannot be ensured, as encryption algorithm cannot meet the stringent timing requirement of 3 ms. Hohlbaum *et al.* [12] highlighted the practical challenges of securing the IEC 61850 message exchanges with IEC 62351-6. The challenge of achieving real time performance for securing the GOOSE and SV with RSA digital signatures were analyzed in resource constrained IEDs running on different platforms. Similarly, Ishchenko and Nuqui [13], evaluated the performance of 1024-bit RSA digital signature algorithm for securing GOOSE message on different platforms. In the literature [12]–[14], all the analysis and evaluations for securing GOOSE messages were based on the RSA digital signatures without specifying the exact signature scheme. However, for securing the GOOSE messages IEC 62351-6 standard explicitly specifies the use of RSA-Probabilistic Signature Scheme based on Signature Scheme with Appendix (RSASSA-PSS), as per RFC 3447, which is also compatible with RFC 2313 (i.e. PKCS version 1.5). This means the signature scheme as per the IEC 62351-6 standard specification for securing GOOSE messages is RSASSA-PKCS1-v1_5. Hence, there is a need for analysis of the RSASSA-PKCS1-v1_5 digital signature scheme's timing performance and evaluate its suitability for securing IEC 61850-based networks as specified by the IEC 62351-6 standard.

Addressing this knowledge gap, this paper presents the performance evaluation and feasibility analysis of RSASSA-PKCS1-v1_5 digital signature scheme. From the results it is concluded that the RSASSA-PKCS1-v1_5 (which is in conformance with both RFC 3447 and RFC 2313) has better performance than the other variants such as RSA (which is not in conformance with either of the RFCs) and RSASSA-PSS (which is in conformance with only RFC 3447). Nevertheless, results show that despite being the fastest algorithm in the pack, RSASSA-PKCS1-v1_5 still cannot be safely implemented in GOOSE messages for time-critical operations. This requires a revision in IEC 62351-6 regarding stipulated signing algorithms.

IEC 62351 is drafted with some recommendations and is revised according to performance evaluation investigations such as the one reported in this manuscript. These findings provide valuable information to experts and researchers in this field, regarding securing GOOSE messages. Building on these findings, novel methods can be developed, and a realistic cybersecurity standard can be drafted for smart grids. This will, in turn, ensure that smart grid communications can be performed in safe and reliable fashion.

The rest of the paper is organized as follows. Section II gives brief introduction to GOOSE protocol followed by explanation of RSASSA-PSS and RSASSA-PKCS1-v1_5 signature schemes. Section III describes the implementation of these algorithms and data collection methodology. Finally, Section IV concludes the paper.

II. OVERVIEW OF RSA SIGNATURE FOR GOOSE MESSAGES

A. GOOSE MESSAGE

GOOSE messages are used to transfer time critical information to coordinate operation in substations. Widely used in protection systems, GOOSE message service ensures fast and reliable communication among substation Intelligent Electronic Devices (IEDs). For this reason, GOOSE messages have strict time requirements. IEC 61850 stipulates that a GOOSE message shall have no more than 3ms delay. Additional time required for implementation of cybersecurity mechanisms shall not cause more delays, including the transmission delays. GOOSE messages can be mapped directly to the ethernet layer, omitting network and transport layer headers and reducing the overall size of message. This, in turn, reduces the propagation and processing delays of the GOOSE messages. Figure 1 shows the structure of an ethernet frame. It includes header information such as destination and source MAC address fields (6 bytes each), ether type field (2 bytes) which represents the protocol to be followed in the data field of ethernet frame, GOOSE PDU (M bytes, depending on the content) and, lastly, a 4-byte Frame Check Sequence (FCS) trailer field. GOOSE PDU consists of APPID, Length, Reserved1, Reserved2, APDU and Extension fields.

Inside the GOOSE PDU, APPID is the application ID for GOOSE (2 bytes) Length field is of 2 bytes which represents the length of rest of the PDU. When M is the total length of GOOSE PDU, M-4 is the value of the length field. Reserved1 field is 2 bytes that contains the value of length of extension field. Reserved2 field is also 2 bytes and stores the 16-bit CRC value. APDU field consists of GOOSE message related information. It is important to note that *Extension field* is used to store the authentication value in case of using digital signatures.

B. IEC 62351-6 RECOMMENDATIONS FOR GOOSE MESSAGE INTEGRITY AND AUTHENTICATION

General procedure to create a digital signature has two steps. Firstly, a hash value is generated for the GOOSE message to be transmitted using hash algorithm such as Secure Hash Algorithm (SHA). Secondly, this hash value is signed by the digital signature to show that a legitimate entity has calculated the reported hash value. IEC 62351-6 stipulates the use of SHA256 to generate hash values. Signing of hash

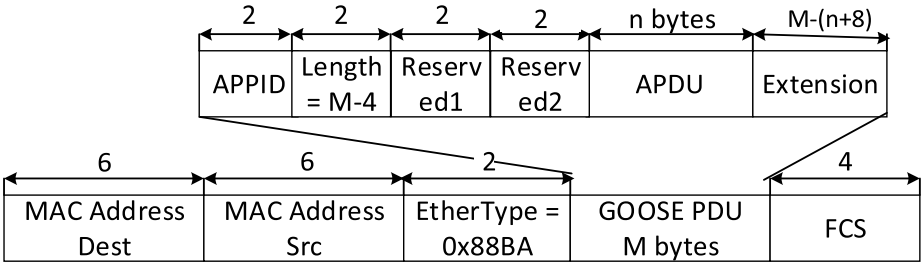

FIGURE 1. GOOSE message format in ethernet layer.

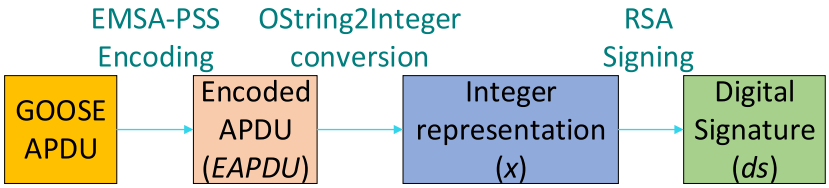

FIGURE 2. Block diagram of RSASSA-PSS signing process.

value involves RSA algorithms which may be implemented in different ways. IEC 62351-6 stipulates RSA-Probabilistic Signature Scheme based on Signature Scheme with Appendix (RSASSA-PSS), as per RFC 3447, which is also compatible with RFC 2313 (i.e. PKCS version 1.5) to achieve integrity and authentication of GOOSE messages. For example, in a 62351 compliant network, when a fault occurs, a protection IED sends a digitally-signed GOOSE message to a breaker IED. Upon receipt, breaker IED regenerates the hash value using SHA algorithm and compares it with the received message. If the process succeeds, then the message is received from a legitimate source and it is not tampered with. Details of this operation can be found in [8].

C. RSASSA-PSS

Probabilistic signature algorithms are more secure than deterministic ones due to additional use of salt value in the signature generation process. Figure 2 is a block diagram that describes the RSASSA-PSS signature generation process. The signature algorithm differs from deterministic algorithms such as Full Domain Hashing Scheme [15]. The signing process takes GOOSE APDU as input and performs signing in three steps. In the first step, GOOSE APDU data is encoded using Encoding Method for Signature Appendix (EMSA) Scheme. The output of the first step is Encoded APDU (EAPDU). The second step is conversion of EAPDU to integer representation (x). The final step is signing of the integer representation. Signing involves encryption with private key of RSA algorithm to generate digital signature (ds). *Algorithm Gen_RSASSA_PSS_*Signature explains RSA signing process to generate a signature. It takes *gooseAPDU* as an input. The public key (*PubKey*) and private key (*PrKey*) pairs are generated using *GenKeys* function. *gooseAPDU* data is encoded using EMSA-PSS encoding (*EMSAEncode*) function that gives encoded APDU data (*EAPDU*). The *EAPDU* is converted to integer representation (x) using *OString2Integer* function. In the last step, the integer representation x is encrypted with the private key (*PrKey*) of RSA using *RSAEnc* function that generates digital signature (*ds*).

The major step in RSASSA-PSS signing process is EMSA-PSS scheme which is illustrated in Figure 3. The scheme takes *gooseAPDU* as input and generates hash value (*gooseAPDUHash*) using SHA256 hash algorithm. The size of the message should not exceed the input limitation of hash function. The result *gooseAPDUHash* is mapped to an Encoded Message (EM) using Mask Generation Function (MGF1). Figure 3 describes the encoding scheme. The first hash algorithm function (SHA256) is used to generate hash value (*gooseAPDUHash*) for the *gooseAPDU*. A random salt value is appended to the combination of concatenated with eight zeros padded *gooseAPDUHash* and eight zeros to form masked hash (*M'*). *M'* is given as input to the second hash algorithm (SHA256). The output of the second hash algorithm is *maskedSeed*. Let *emLen* be the length of the final EM, *hLen* be the length of the generated hash value (*gooseAPDUHash*) and *sLen* be the length of the salt value, then EM length should be at least the length of hash value plus length of salt value plus 2.

$$emLen > hLen + sLen + 2 \tag{1}$$

Salt value is added for randomization. Randomization causes output to be different for equivalent inputs to prevent dictionary attacks. The output of the second hash function is *maskedSeed* which is an input to the MGF1 along with *maskLen* value. The length of the *maskedSeed* is also *hLen*. *maskLen* value is the intended length of the output of the MGF1. Its value is equal to *emLen-hLen*-1. The output of the MGF1 is *dbMask* which is of *maskLen* size. At this stage, *PS* value is generated which consists of repeated zeros as shown in Figure 3. The length of the *PS* value is

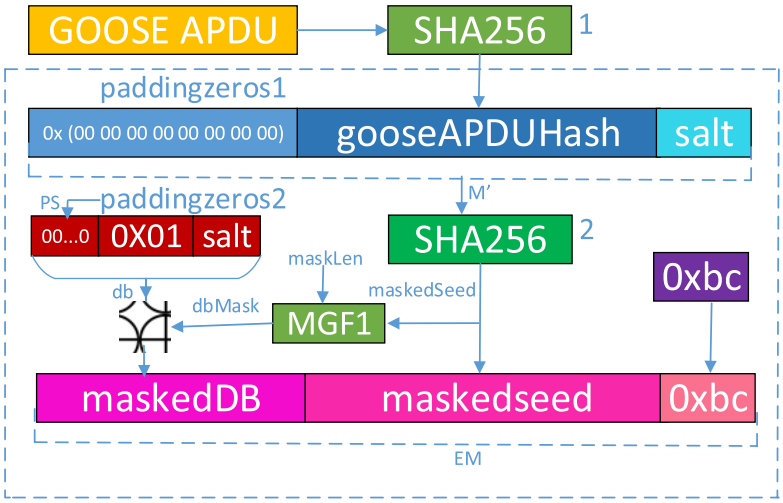

FIGURE 3. EMSA-PSS Encoding process.

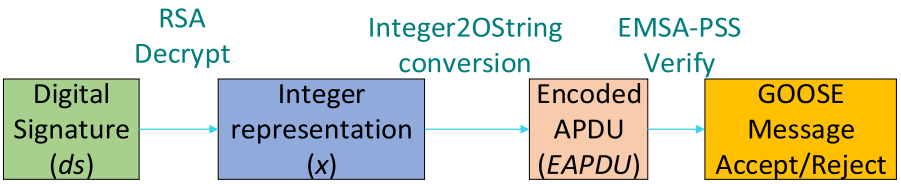

FIGURE 4. Block Diagram of RSASSA-PSS verification process.

Algorithm 1 Gen_RSASSA_PSS_Signature(*gooseAPDU*)

1: *(PubKey, PrKey)* ← *GenKeys()*
2: *EAPDU* ← *EMSAEncode(gooseAPDU)*
3: *x* ←*OString2Integer(EAPDU)*
4: *ds* ← $RSAEnc_{PrKey}(x)$ *//RSASigning.*
5: *return ds*

Algorithm 2 Verify_RSASSA_PSS_Signature(gooseAPDU, ds)

1: *x* ← $RSADec_{PubKey}$ *(ds)*
2: *EAPDU*← *Integer2OString(x)*
3: *if (EMSAPSSVerify(gooseAPDU, EAPDU)) then*
4: *GOOSE Message is valid Accept.*
5: *return True.*
else
6: *GOOSE Message is invalid Reject.*
7: *return False.*

emLen-hLen-sLen-2. *PS* value is concatenated in sequence with hexadecimal value 0x01 and salt value. The resultant concatenation is *DB*. *DB* length is *emLen-hLen-1*. The output of MGF1 (*dbMask*) is XORed with *DB* value to generate *maskedDB* which is of length *emLen-hLen-1*. Let *emBits* be the maximal allowed bit length for integer representation. The *emBits* value is used in *String2Integer* Conversion function. Set the *8emLen – emBits* bits of leftmost octet of the *maskedDB* to zero before final concatenation with *maskedSeed*. Finally, concatenation of *maskedDB* value, *maskedSeed* and one-byte hexadecimal 0xbc value for compatibility purpose gives the *EM*. The encoded string is converted to integer representation using *String2Integer* conversion function which is encrypted (*RSAEnc*) by private key (*PrKey*) of RSA algorithm that generates digital signature (*ds*). Generated *ds* and concatenated with *gooseAPDU* are sent to the receiver.

At the receiver end, verification operation follows reverse steps to the signing process as given in *Verify_RSASSA_PSS_Signature* algorithm. It recovers salt value, then recomputes the hash value *H* and compares it. Initially, received ds is decrypted with *PubKey* of RSA algorithm using RSA Decryption function (*RSADec*). The output of *RSADec* is integer representation (*x*). It is converted to octet string format using *Integer2OString* conversion function. Further, the output of the conversion function (*EAPDU*) is verified using *EMSAPSSVerify* function to check the consistency with the received GOOSE message (*gooseAPDU*). Figure 4 shows the block diagram of verification process of RSASSA-PSS.

The EMSA-PSS Verifying scheme follows reverse steps to the encoding scheme to recover salt value then recomputes the hash value (*maskSeed*) and compares it. Figure 5 shows the flow chart of EMSA-PSS verification process.

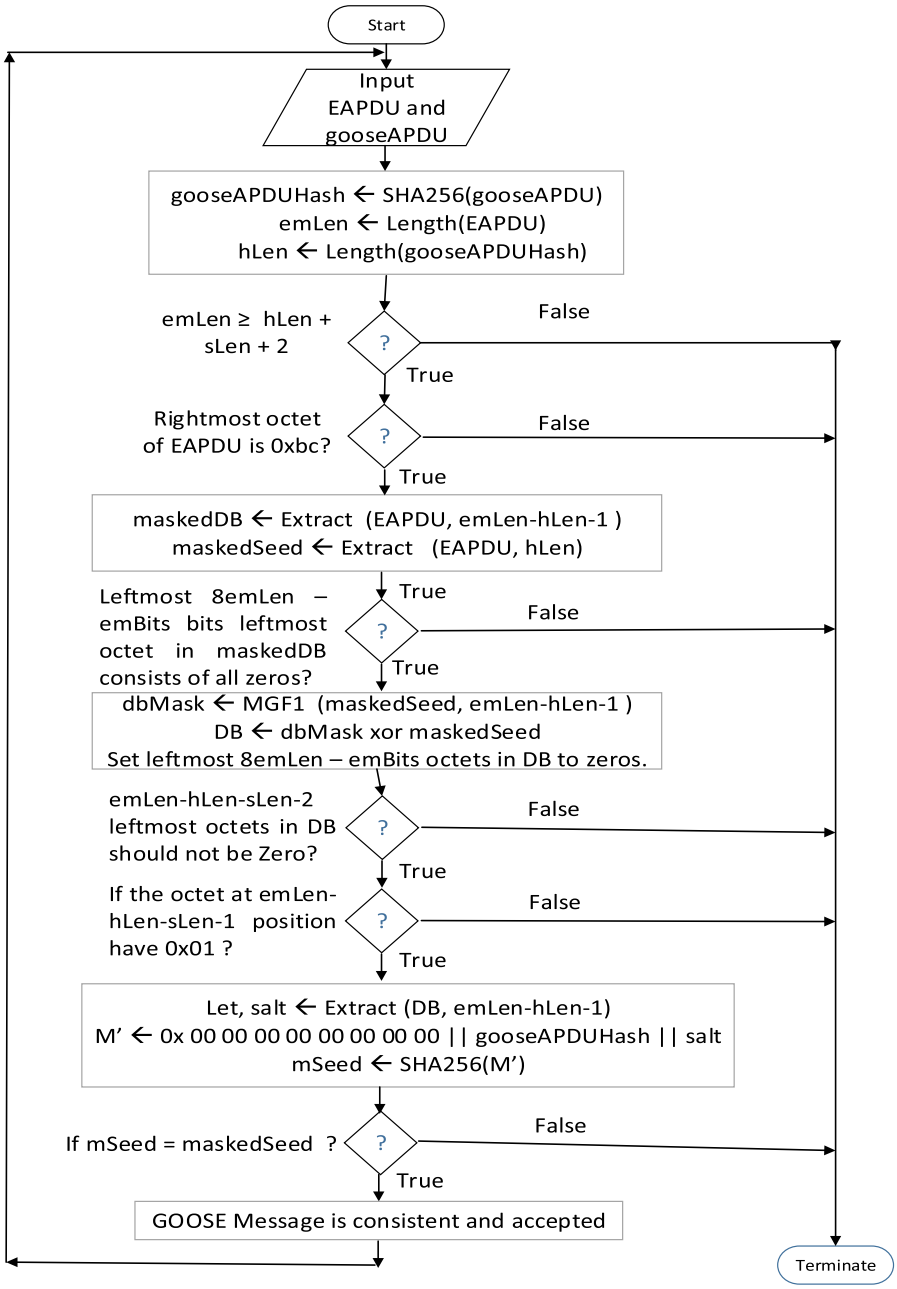

FIGURE 5. Flow chart of EMSA-PSS verification.

EMSA-PSS verifying scheme is as follows: initially *gooseAPDUHash* is generated with the received *gooseAPDU* message. The length of the encoded string (*EAPDU*) is checked. If the length is less than $hLen + sLen + 2$, then it is inconsistent. If the right most value of EAPDU message does not have hexadecimal 0xbc, then it is inconsistent. If these two parameters are consistent, the process continues to extract the *maskedDB* and *maskedSeed* values with their lengths from *EAPDU*. Leftmost *emLen-hLen-1* octet length is *maskedDB* followed by *maskedSeed* with the length of *hLen*. If the leftmost *8emLen – emBits* of the *maskedDB* are not zeros, then *EAPDU* is inconsistent. As *maskedSeed* and the length of the *maskedDB* value are known, *dbMask* value can be generated using *MGF1*. When *dbMask* value is XORed with *maskedDB* to get *DB* value. Leftmost *8emLen – emBits* value in *DB* to zeros. It is possible to check whether leftmost *emLen-hLen-sLen-2* octets (*ps* value) in *DB* consists of zeros and *emLen-hLen-sLen-1* leftmost position of *DB* value is 0x01. If the check fails, then *EAPDU* is inconsistent. It is possible to extract salt value from *DB* value. Once salt value is obtained, new *maskedSeed* (*mSeed*) can be calculated with the combine inputs of eight zeros, *gooseAPDUhash* and

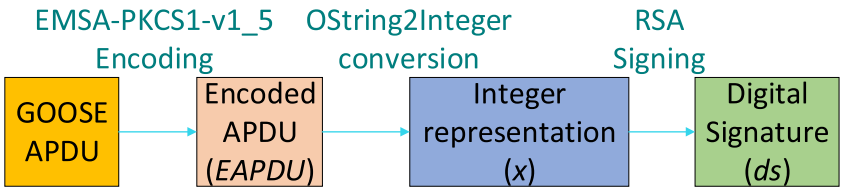

FIGURE 6. Block diagram of RSASSA–PKCSV1_5 signature generation process.

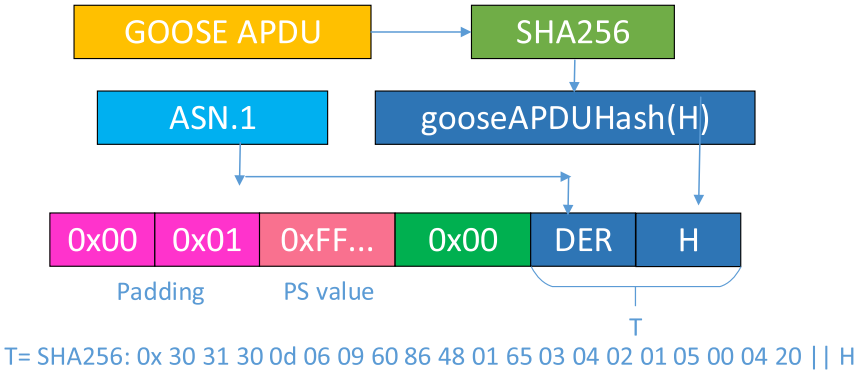

FIGURE 7. EMSA-PSS Encoding process.

Algorithm 3 Gen_RSASSA_PKCS1-v1_5_Signature (goosePDU)

1: *(PubKey, PrKey)* ← *GenKeys()*
3: *EAPDU* ← *EMSA_PKCS1-v1_5_Encode (gooseAPDU)*
4: x ← *OString2Integer(EAPDU)*
5: *ds* ← $RSAEnc_{PrKey}$ *(x) //RSASigning.*
6: *return ds*

Algorithm 4 Verify_RSASSA_PKCS1-v1_5_Signature (gooseAPDU, ds)

1: x ← $RSADec_{PubKey}$ *(ds)*
2: *EAPDU* ← *Integer2OString(x)*
3: *EAPDU'* ← *EMSA_PKCS1-v1_5_Encode (gooseAPDU)*
4: *if (EAPDU = EPADU') then*
4: *GOOSE Message is valid Accept.*
5: *return True.*
else
6: *GOOSE Message is invalid Reject.*
7: *return False.*

salt value with the hash algorithm SHA256. By comparing the extracted *maskedSeed* from *EAPDU* with the regenerated *maskedSeed* (*M'*), the consistency of the GOOSE message can be verified.

D. RSASSA-PKCS-v15

RSASSA-PKCS1-v1_5 algorithm makes use of EMSA-PKCS1-v1_5 encoding scheme to generate a digital signature. Figure 6 shows the block diagram of RSASSA-PKCS1v1_5 signature generation process. It describes the signature generation process of RSASSA-PKCS1-v1_5 (Public Key Cryptography Standard #1 version 1.5) digital signature algorithm. The signing process takes *gooseAPDU* as input and performs signing in three steps. In the first step, *gooseAPDU* is encoded using Encoding Method for Signature Appendix (EMSA-PKCS1-v1_5) Scheme. The second step is conversion of encoded string to integer format. The final step is signing of the integer representation. Signing involves encryption with private key of RSA algorithm to generate the digital signature.

Algorithm *Gen_RSASSA_PKCS1-v1_5_Signature* explains RSA signing process. The public key (*PubKey*) and private key (*PrKey*) pairs are generated using *GenKeys* function *gooseAPDU* data is inputted to the algorithm. *gooseAPDU* data is encoded using EMSA-PKCS1-v1_5 (EMSA_PKCS1-v1_5_Encode) encoding function. The generated octet string of encoded value (*EAPDU*) is converted to integer representation (x) for encrypting with the private key of RSA using *RSAEnc* function.

EMSA-PKCS1-v1_5 scheme is illustrated in the Figure 7. The scheme takes *gooseAPDU* as input and generates hash value *gooseAPDUHash* (H) using SHA256 algorithm. The size of the input message should not exceed the input limitation of hash function. The output H is padded with additional bytes which are used in verification process. The padded bytes are 0x00, 0x01, *ps* value, 0x00 and DER Encoding

TABLE 1. Computational times for digital signature algorithm.

Digital Signature Algorithm	Key Size (bits)	Digital Signature Signing time (ms)	Digital Signature verification time (ms)	Processor	Reference
RSASSA-PKCS1-v1_5	1024	0.942	0.283	Intel i5-3210M CPU @ 2.50GHz	Work presented in this paper.
	2048	3.56	0.75		
RSASSA-PSS	1024	5.2	0.25		
	2048	6.1	0.28		
RSA	1024	6	4		
	2048	10	7		
RSA	1024	6.8	-	Pentium M 1.7 GHz	[12]
RSA	1024	4	-	Intel Core 2 Duo @ 2.2 GHz	
RSA	1024	3.748	0.155	FPGA (100 MHz)	
	1024	1.917	0.129	FPGA (200 MHz)	
RSA	1024	0.3		Xeon server 2.53GHz quad-core	[13]
RSA	1024	>6	-	BeagleBone Black (TIAM3359 ARM Cortex A8 CPU @ 1 GHz)	
RSA	1024	>10	-	Broadcom BCM2836 quad-core ARM Cortex A7 overclocked at 1 GHz	

of hash function and hash value (ASN.1) as shown in the Figure 7. DER specifies digest information which consists of identifier of hash function to be used and hash value generated by the hash function. The hash function SHA256 should be used according to IEC 62351-6 standard. Let the output encoded message length be *emLen*. The *ps* value consists of repetition of 0xff. The length of *ps* value is determined by the length of the hash value *hLen* generated by the hash function (SHA256) which is defined by DER. The length of *ps* value is,

$$Ps = mLen - hLen - 3, \quad where\ emLen < hLen + 11 \tag{2}$$

The final encoded message is a concatenation of padded bytes, *ps* value and *T* value which consists DER encoding plus hash value (H) as shown in Figure 7. Verification process follows the same direction. It generates a new encoded message and compares it with the received encoded message. If both are consistent, then the signature is valid; otherwise, the GOOSE message is discarded. Algorithm *Verify_RSASSA_PKCS1-v1_5_Signature* takes *goosePDU* and *ds* value as inputs. The *ds* value is decrypted using *PubKey* of RSA algorithm. The output is an integer representation x, which is converted to octet string using *Integer2OString* function. The output of *Integer2OString* function is *EAPDU*. A new *EAPDU* (*EAPDU'*) can be generated with the *goosePDU* and compared it with the *EAPDU*. If both are consistent, GOOSE message is accepted; otherwise, it is discarded.

III. IMPLEMENTATION AND ANALYSIS OF RSA DIGITAL SIGNATURE ALGORITHMS FOR GOOSE MESSAGES

In this section, operation times of different RSA digital signature variants are calculated. These results are analyzed to evaluate their applicability to GOOSE messages. Table 1 shows the key size, signing and verification times as well as the platform that is used. In this paper, the platform selected for evaluation is a system with Intel®i5-3210M CPU processor with 4 GB RAM. If this system can meet timing requirements, then current IEDs should not face any difficulties as they have much higher computing power (Intel®Core i7-3555LE with 8 GB RAM) [16].

Further, the results are compared with the existing literature [12], [13]. Hohlbaum *et al.* [12] measured digital signature computation time in three different platforms: Pentium M, Core 2 Duo and FPGA. None of these platforms has satisfactory performance for real time requirements of GOOSE messages. Ishchenko and Nuqui [13] generated the results in 2.53GHz quad-core Xeon server CPU, BeagleBone Black with TIAM3359 ARM Cortex A8 CPU at 1 GHz and Raspberry Pi 2 (RPi2) with a Broadcom BCM2836 quad-core ARM Cortex A7 overclocked at 1 GHz.

Xeon server CPU requires 0.3 milliseconds to complete one RSA 1024-bit computation due to its inherent hardware. However, the server processors like Xeon are not used in IEDs and it is not reasonable to expect that will change soon.

The RSASSA-PSS schemes resulted in overall computational times of 5.45 ms and 6.38 ms for 1024- and 2048-bit key sizes respectively. The computational times are much greater than the 3 ms time requirement for GOOSE message. Hence, the RSASSA-PSS scheme is found to be unsuitable for securing GOOSE messages. The results show that RSASSA-PKCS1-v1_5, stipulated by IEC 62351-6, has better performance than other RSA schemes. However, still the fastest variant, i.e. RSASSA-PKCS v1_5 with 1024 key size, has a computational time of 1.225 ms. This corresponds to 40% of the 3 ms maximum delay allowed for GOOSE messages. The RSASSA-PKCS v1_5 with 2048 key size results in computational times more than 4 ms which is unacceptable.

The legacy IEDs with slower processors may result in higher delays which will impact the performance of the system. Also, starting from 2013, 1024 key size for RSA algorithms is not allowed by NIST. Therefore, despite having better performance compared with other RSA schemes, the IEC 62351-6 stipulated RSASSA-PKCS1-v1_5 will not be suitable for securing GOOSE messages. Hence, there is a need to revise the IEC 62351-6 standard. A new authentication scheme has to be developed that meets requirements of both the cybersecurity and timing domains.

IV. CONCLUSIONS

Security in power system control operations is becoming increasingly important due to its reliance information infrastructure. IEC 61850 standard specifies protocols to solve the problem of interoperability. Of these protocols, GOOSE is specifically designed to meet the needs of operations with very strict timing requirements. IEC 62351-6 standard complements IEC 61850 standard and specifies security profiles to protect IEC 61850 message exchanges. To achieve integrity and authentication of GOOSE messages, IEC 62351-6 standard specifies the use of RSASSA-PKCS1-v1_5 signature scheme. It is important that these cybersecurity mechanisms do not cause delays in of GOOSE messages more than the allowed limits.

For this reason, timing performance of the RSASSA-PKCS1-v1_5 digital signature algorithm and compared it with other existing schemes reported in literature. Despite showing the best performance, even RSASSA-PKCS1-v1_5 does not meet timing requirements of GOOSE messages. Therefore, there is a need for revising IEC 62351-6 with these considerations. A new authentication scheme has to be developed that meets requirements of both the cybersecurity and timing domains. Considering its speed, Message Authentication Codes (MAC) based algorithms are a good candidate for securing GOOSE messages.

REFERENCES

[1] I. Ali, S. M. S. Hussain, A. Tak, and T. S. Ustun, "Communication modeling for differential protection in IEC-61850-based substations," *IEEE Trans. Ind. Appl.*, vol. 54, no. 1, pp. 135–142, Jan./Feb. 2018.
[2] M. A. Aftab, S. Roostaee, S. M. Suhail Hussain, I. Ali, M. S. Thomas, and S. Mehfuz, "Performance evaluation of IEC 61850 GOOSE-based inter-substation communication for accelerated distance protection scheme," *IET Gener., Transmiss. Distrib.*, vol. 12, no. 18, pp. 4089–4098, Oct. 2018.
[3] *Communication Networks and Systems for Power Utility Automation, 2.0.*, Standard IEC 61850, IEC, 2013.
[4] J. H. Hong, C.-C. Liu, and M. Govindarasu, "Integrated anomaly detection for cyber security of the substations," *IEEE Trans. Smart Grid*, vol. 5, no. 4, pp. 1643–1653, Jul. 2014.
[5] A. Chattopadhyay, A. Ukil, D. Jap, and S. Bhasin, "Toward threat of implementation attacks on substation security: Case study on fault detection and isolation," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2442–2451, Jun. 2018.
[6] M. T. A. Rashid, S. Yussof, Y. Yusoff, and R. Ismail, "A review of security attacks on IEC61850 substation automation system network," in *Proc. 6th Int. Conf. Inf. Technol. Multimedia*, Putrajaya, Malaysia, 2014, pp. 5–10.
[7] C. Diago and A. Forshaw, "Cybersecurity for shared infrastructure substation networks with IEC 61850 GOOSE and sampled values," *J. Eng.*, vol. 2018, no. 15, pp. 1195–1198, Oct. 2018.
[8] S. M. Farooq, S. M. S. Hussain, S. Kiran, and T. S. Ustun, "Certificate based authentication mechanism for PMU communication networks based on IEC 61850-90-5," *Electronics*, vol. 7, p. 370, Dec. 2018.
[9] *Power Systems Management and Associated Information Exchange—Data and Communications Security—Part 6: Security for IEC 61850 1.0.*, Standard IEC 62351-6, IEC, 2007.
[10] *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, document RFC 3447, IETF, Feb. 2003.
[11] *Public-Key Cryptography Standards (PKCS) #1: RSA Encryption Version 1.5*, document RFC 2313, IETF, Mar. 1998.
[12] F. Hohlbaum, M. Braendle, and A. Fernando, "Cyber security practical considerations for implementing IEC 62351," in *Proc. PAC World Conf.*, Dublin, Republic of Ireland, Jun. 2010, pp. 21–24.
[13] D. Ishchenko and R. Nuqui, "Secure communication of intelligent electronic devices in digital substations," in *Proc. IEEE/PES Transmiss. Distrib. Conf. Expo. (T&D)*, Denver, CO, USA, Apr. 2018, pp. 1–5.
[14] W. Fangfang, W. Huazhong, C. Dongqing, and P. Yong, "Substation communication security research based on hybrid encryption of DES and RSA," in *Proc. 9th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Beijing, China, 2013, pp. 437–441.
[15] M. Bellare and P. Rogaway, "The exact security of digital signatures-how to sign with RSA and Rabin," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1070. Berlin, Germany: Springer, 1996.
[16] *Data Sheet-SEL 3555 Real Time Automation Controller (RTAC)*. Accessed: Aug. 3, 2019. [Online]. Available: https://cdn.selinc.com/assets/Literature/Product%20Literature/Data%20Sheets/3555_DS_20181207.pdf?v=20181231-170401

SHAIK MULLAPATHI FAROOQ received the B.Tech. and M.Tech. degrees in computer science and engineering from Jawaharlal Nehru Technological University, Hyderabad, India. He is currently pursuing the Ph.D. degree in computer science engineering with Yogi Vemana University, Kadapa, India. He was a Visiting Researcher with the Fukushima Renewable Energy Institute, AIST (FREA), Japan, in 2018. His research interests include cryptography, cyber physical systems, and cybersecurity in vehicular networks and power systems.

S. M. SUHAIL HUSSAIN received the Ph.D. degree in electrical engineering from Jamia Millia Islamia (a Central University), New Delhi, India, in 2018. He is currently a Postdoctoral Researcher with the Fukushima Renewable Energy Institute, AIST (FREA), Koriyama, Japan. His research interests include power system communications, smart grid, cyber-physical systems, cybersecurity in smart grids, IEC 61850, and renewable energy integration. He was a recipient of the IEEE Standards Education Grant approved by the IEEE Standards Education Committee for implementing the project and submitting a student application paper in 2014–2015.

TAHA SELIM USTUN (M'04) received the Ph.D. degree in electrical engineering from Victoria University, Melbourne, VIC, Australia. He is currently a Researcher with the Fukushima Renewable Energy Institute, AIST (FREA), and leads the Smart Grid Cybersecurity Lab. Prior to that, he was an Assistant Professor of electrical engineering with the School of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include power systems protection, communication in power networks, distributed generation, microgrids, electric vehicle integration, and cybersecurity in smart grids. He is a member of the IEEE 2800 Working Groups and the IEC Renewable Energy Management Working Group 8. He has edited several books and special issues with international publishing houses. He is a Reviewer in reputable journals and has taken active roles in organizing international conferences and chairing sessions. He has been invited to run specialist courses in Africa, India, and China. He delivered talks for Qatar Foundation, World Energy Council, Waterloo Global Science Initiative, and European Union Energy Initiative (EUEI).