Structural Weaknesses in the Open Smart Grid Protocol

Klaus Kursawe Christiane Peters*

European Network for Cyber Security
The Hague, The Netherlands,
{klaus.kursawe,christiane.peters}@encs.eu

Abstract

The Open Smart Grid Protocol (OSGP) is currently deployed in various European countries in large-scale Smart Metering projects. The protocol was developed by the OSGP Alliance and published as a standard by the European Telecommunications Standards Institute (ETSI).

We identify several security issues in the OSG Protocol, primarily the use of a weak digest function and the way the protocol utilizes the RC4 algorithm for encryption. A straight-forward oracle attack triggers the leakage of key material of the digest function. We outline how an attacker can make use of the simple protocol structure to send maliciously altered messages with valid authentication tags to the meters.

The results of our analysis have been made available to the manufacturer the beginning of 2014, and mitigation strategies have been discussed with the vendor and utilities.

Keywords: protocol analysis, authentication protocol, OSGP, ISO/IEC 14908, Advanced Smart Metering, RC4, oracle attack, bit-flipping attack

1 Introduction

The Open Smart Grid Protocol (OSGP) [10] aims at securing powerline communication (PLC) between smart meters and data concentrators. A data concentrator collects data from potentially several hundred meters in a PLC segment and forwards this data to the head end.

The OSGP resides in the application layer of the protocol stack defined by the ISO/IEC 14908 standard [14], short 'EN 14908'. The OSGP is primarily used for smart-metering applications, however it was designed for a wider usage in smart grid devices. The protocol stack is very lightweight, at the price of avoiding NIST recommended cryptographic primitives (e.g., the Advanced Encryption Standard (AES) in an authenticated mode) in favor of less computationally intense ones: the RC4 stream cipher for encryption and a non-standard digest function for message authentication.

We found that the biggest issue is the combination of a stream cipher with a mostly linear digest function, opening the possibility for attacks on cryptographic keys and messages. One can for example exploit this by using the receiver of a message as an 'oracle'. Given a properly encrypted and authenticated message, an attacker guesses a message/key bit, sends a specially crafted message to the receiver, and uses the device response — either a rejection

*This work was supported by the European Commission FP7 Programme under Contract No. 607109 (SEGRID). Date: 2015.02.05.

or an acceptance of the message — to confirm or reject the guess. As this can be done for individual bits, an attacker can reconstruct the entire encryption key with at most 96×3 message responses.

Overall, we identified four potential weaknesses in the OSG Protocol:

- **Use of RC4.** Over the last years a number of weaknesses have been identified in the RC4 stream cipher. One of the issues is a correlation between the key and the RC4 keystream that can be used to reconstruct the key. This weakness was successfully exploited using chosen initialization vectors (IVs) to break the WEP (Wired Equivalent Protection) standard which makes heavy use of RC4. The attacks [18, 28] break WEP within seconds and attack tools such as Aircrack-ng [1] are freely available and a standard tool for penetration testers [17].

  The use of RC4 in the OSGP is similar to the way RC4 was used in the WEP standard. While a new key is generated per OSGP message, only the first 8 bytes of this key are in fact different from each other, while the remaining 8 bytes are constant. Furthermore, the initial 8 bytes are a constant xored with a value that is publicly known. While the authors are not aware of a concrete attack on this, it does imply a strong correlation between the used keys, and supplies an attacker with rich data to analyze. Furthermore, the setting is rather close to the one used in the WEP standard, which means there is a large body of experience in the field on statistical key recovery.

  Another issue with RC4 in the OSGP is that it is used with weak authentication. Given the nature of a stream cipher, an attacker who gets hold of a ciphertext can alter bits in chosen positions. Using the knowledge of the message space of the EN 14908 standard an attacker can send altered messages that are valid (with high probability). Moreover, she can authenticate these messages exploiting the weaknesses of the digest function that are described in the following.

- **Weak digest function.** The digest function used in the OSG Protocol for message authentication is almost linear and is implemented in a way that not only disables authentication but can also decrypt an intercepted message in time linear in the message length. The structural weaknesses in the digest function allow for a 'meet-in-the-middle' attack (effectively halving the key size for a brute-force attack on the key), modification of messages in a controlled way, recomputing the correct digest without need for encryption- or authentication keys, and using a device's NACK messages[1] on a wrong digest to reconstruct both authentication key and message.

- **Undefined broadcast security.** While the broadcast function is called 'Secure Broadcast', there is very little security specified for it. This is particularly worrying as this mechanism is used to send firmware updates, which are distributed using the secure broadcast mechanism without any specified measures to provide source authentication.

- **Key usage.** While the protocol does use session keys for encryption, it uses one master key for authentication. However, this authentication key is used to derive the encryption session keys. Thus if the authentication key is compromised, all session keys are known to the attacker. As the authentication key is used with a weak authentication algorithm, it is fairly exposed to a number of attacks, and a compromise of this key is well possible.

[1] A message containing a **N**egative **A**cknowledgement **Ch**aracter to indicate an error.

Exploiting the structural weaknesses. These structural weaknesses can be exploited in various ways. The attacks we described in this paper are not overly sophisticated, and should rather serve as an illustration of how weak cryptographic building blocks can enable an attacker to potentially hijack a whole PLC segment. A relatively simple attack breaks the OSGP digest function in a way that an attacker can not only forge messages, but — given access to one sufficiently long authentic message — can reconstruct cryptographic keys with low effort.

We describe how to recover the key which then can be used to authenticate modified messages. Due to the usage of the RC4 stream cipher and the simple message structure it is possible to send false commands that are properly authenticated using the OSGP digest. While we have not implemented the attack yet and there might be (or even, likely are) practical issues to overcome, the goal of our analysis was not to provide a readily usable attack tool, but to demonstrate the structural weaknesses of the used building blocks. These should be seen like cracks in a dam — a last warning sign that something needs to be fixed before the real damage has been done.

More recent work. The results of our analysis have been made available to the manufacturer the beginning of 2014, and mitigation strategies have been discussed with the vendor and utilities. Due to the criticality of the weaknesses, we have held our paper so far under embargo.

Recently, Jovanovich and Neves independently analyzed the OSGP [16] and kindly shared their results before publication. They identified the same issues focusing mainly on the digest function. Their differential analysis of the digest function goes deeper than ours, rendering exploits of the OSGP weaknesses even more practical. We have planned a more thorough comparison of their and our results as future work.

Organization of the paper. In Section 2 we describe the general setup of the OSG Protocol and identify first issues. In Section 3 we discuss the usage of the RC4 stream cipher in the OSGP and potential attacks. Section 4 introduces the OSGP digest function and notation. Section 5 outlines an attack on the digest function that uses the meter as an oracle. Section 6 discusses how to make meet-in-the-middle attacks effective using the structure of the digest function. Section 7 provides lessons learned and advice how to replace the weak primitives by recognized cryptographic primitives in the standard while respecting the constraints of the PLC environment.

2 Usage of the OSGP

In this section we describe the protocol according to the specifications in the "Group specification GS OSG 001: Open Smart Grid Protocol" by the ETSI Open Smart Grid Industry Specification Group [10]. We outline the problems of using RC4 and the weak digest function for encryption and authentication in the OSGP.

The EN 14908 standard [14] defines communication for layers 2 to 7 in control networks. Applications lie amongst others in the field of smart metering, specifically in the communication between a data concentrator and smart meters in one PLC segment. Such a segment contains meters in a whole neighborhood, possibly several hundred devices. Meters can act as repeaters to forward information to the targeted devices. The data concentrator acts as master with connected connected proxies and meters as slaves.

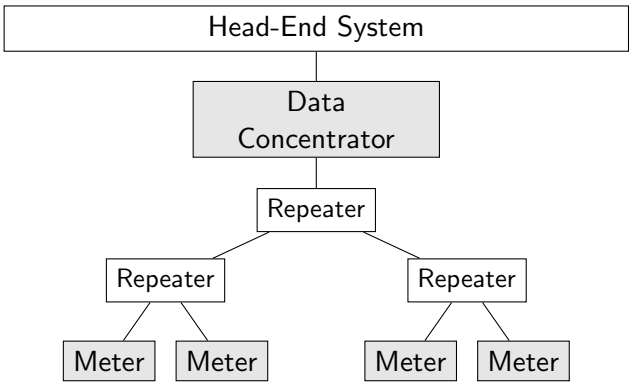

Figure 1: Head-end connected to PLC segment with data concentrator (DC), meters acting as repeaters, and meters as end nodes.

Application layer security. The OSGP was specifically designed to secure the application layer in smart meter communications. EN 14908 claims to secure the transport and session layers. Applications use the EN 14908 communication protocol at layers 2 to 6 to facilitate the lower-layer communication between connected nodes and use the OSGP at layer 7 to provide message authentication and encryption. The OSGP digest function is supposed to provide message authentication. For encryption the RC4 stream cipher is used. Encryption keys are generated using the EN 14908 'encryption function' which we will discuss in the next paragraph.

EN 14908 legacy. The EN 14908 defines security only at the transport layer. It specifies an 'encryption function' that is almost identical to the OSGP digest function. Section 11.12 in [14] admits that "The encryption algorithm [..] facilitates one way encoding rather than real encryption." The randomness requirements in EN 14908 do not adhere to the state of the art in cryptographic randomness. Moreover, "Any 48-bit number is a valid encryption key." In the sample implementation random numbers are generated using the `rand()` function. Needless to say, this function is not considered secure for cryptographic use, and should be replaced.

The main difference is that the OSGP digest function is initialized with an initial state of 0, while the EN 14908 encryption function receives a random value as a parameter to use for initial setup. Another difference is that the EN 14908 encryption function is used as a challenge-response protocol. Here the challenger sends a message and a random value, and receives back an encrypted and authenticated response message. This makes an attack rather easy, as the challenger has full freedom to choose which messages the server encrypts, and can then investigate the outcome. The challenger would then reuse the same randomness with an altered message as will be described in Section 5, and thus recover the keys from there.

Message structure. The OSGP inherits the message structure from EN 14908. The first one or two bytes of the APDU (Application Protocol Data Unit) define the destination and type of the message. The remaining bytes constitute the (possibly encrypted) message bytes and a digest.

Commands are organized in tables that are stored on the devices. A data concentrator initiates the interaction with connected meters by sending the addresses of the commands.

The meter looks up the address in the corresponding table, executes the command associated with the table entry, and writes the requested results in a transaction-response table which is then sent back to the data concentrator.

Section 9.2 of the OSGP standard limits the buffer to a total of 114 bytes. When sending a read request, the table specifying the various read commands is limited to 84 bytes; response tables (writes) are limited to 75 bytes.

Device keys. Each OSGP device is manufactured with a unique symmetric key, called the Open Media Access Key (OMA key). This key can and is typically updated by the data concentrator when the device is placed in the field. This 96-bit OMA key is used for authentication of OSGP messages. It is also used for authentication of certain EN 14908 messages. Moreover, the OSGP derives encryption keys from the OMA key.

For encryption a 128-bit 'Base Encryption Key' (BEK) is set up as follows: the device encrypts the two fixed patterns `81 3F 52 9A 7B E3 89 BA` and `72 B0 91 8D 44 05 AA 57`, using the EN 14908 encryption function using the current OMA key. The resulting two digests of each 8 bytes are concatenated to form the BEK. If a malicious server manages to construct a request of the form of those two patterns it can challenge the sender with 0 and thus get half of the bytes of the base key as a response. Also note that the entropy of this 128-bit key are only 96 bits derived from the OMA key.

Encryption and decryption of OSGP messages. The general approach used in the OSGP for message authentication of encrypted messages is "encrypt-and-authenticate", a method which has been shown to be generically insecure [19]. However, encryption and authentication are not completely independent operations. As shown in Figure 2, first a digest is computed on the plaintext message; then the plaintext is encrypted using the RC4 stream cipher with a session key derived from the digest. In particular, the sender sets up a session key by xoring the 8-byte digest of the request message to the first 8 bytes of the BEK of the receiving device. The receiver takes the digest of the request message and computes the session key in the same way. We will discuss in the following how this construction shows substantial structural similarities to the way RC4 keys were set up in the WEP standard, and how this might allow for attacks similar to the ones WEP is already experiencing.

Updating keys. The EN 14908 uses an *increment key* function that seems rather weak, adding (without carry) a 6-byte key to the current key; a better idea would be to take the old and the new key into a hash function to derive the actual new key. In either case, once a key is compromised, or partial information about the key leaks, updating the key does not help at all.

Secure broadcast and firmware updates. The specification is somewhat unclear on the protection properties of the secure broadcast protocol; no authentication or encryption is defined in the specifications, and the basic protocols are not directly applicable, as they (correctly) require every message to add the receiver identity into the authenticated part.

In either case, there appears to be some risk here. As every OSGP enabled meter may act as a repeater, it is relatively easy for an attacker to get legit access to the broadcast medium, and it appears reasonably easy to obtain the valid broadcast sequence number. As the *secure broadcast* is also used for firmware updates, this can lead to a somewhat dangerous situation given that the standard does not define source authentication by means of digital signatures.

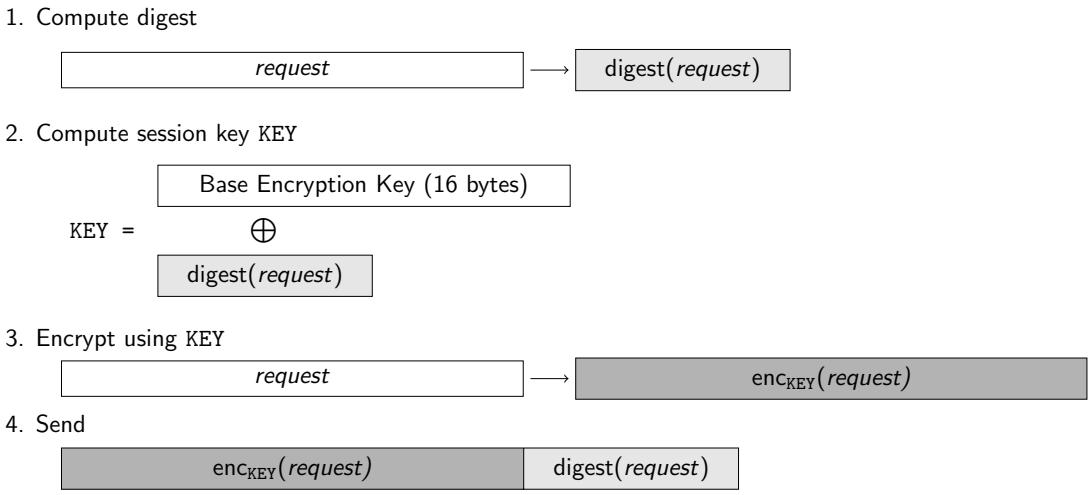

Figure 2: **Encryption and authentication process.** First the digest is computed on the request. Then encryption key is derived from the digest by xoring the digest to the first 8 bytes of the Base Encryption Key. Then the request is encrypted under this key. Ciphertext and digest are concatenated to form the message.

3 RC4 in the OSGP

The RC4 stream cipher was developed by Ron Rivest at the end of the 1980s. The first biases in the keystream were identified in the 1990s [25] and since then cryptanalysts have published a stream of papers identifying flaws in the RC4 design [12, 11, 21, 20, 26, 27, 29]. In the last years more and more real-world attacks on applications using RC4 for encryption were carried out successfully, showing that RC4 is in desperate need of replacement [28, 2, 24]. An overview of the RC4 weaknesses can be found in [13]. Large companies such as Microsoft and Cisco strongly advise against the usage of RC4 and are currently phasing out code containing RC4 [23, 6].

The OSGP uses RC4 to encrypt certain messages. Given that not all messages are encrypted, one can assume that the encrypted messages contain critical commands and sensitive data.

Why stream ciphers should be avoided in control systems. The usage of a stream cipher for communication in a control network is rather unusual. While stream ciphers are designed to encrypt large bulks of data in a very efficient way, stream ciphers are rarely used to encrypt a set of messages of only a couple of bytes. An additional problem when deploying stream ciphers is that they are usually relatively slow in setting up a new key, so the efficiency gained by stream ciphers being comparatively simple vanish quickly if a lot of small messages need to be encrypted with different keys.

Without proper authentication ciphertexts generated by stream ciphers are vulnerable to bit-flipping attacks. Without proper initialization stream ciphers are likely to have statistical biases in their keystream.

Bit-flipping attack on OSGP messages. As opposed to a block cipher, where a change in a single bit of the ciphertext affects the entire block (i.e., changing one ciphertext bit changes

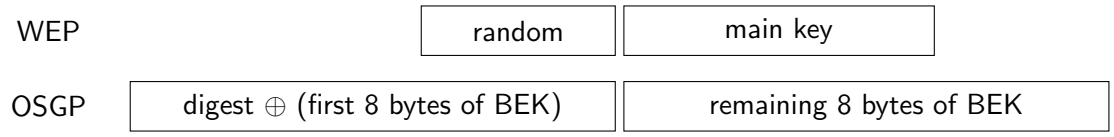

Figure 3: **Initializing vectors in WEP and OSGP.**

half of the plaintext bits in the corresponding block), in a stream cipher an attacker has more control. Changing one bit in the ciphertext flips exactly the same bit in the plaintext, allowing an attacker to selectively flip a bit, even if she does not know if she changed a 0 to 1 or the other way around. This means that a stream cipher needs to be combined with a strong method for message authentication, as it is straightforward to perform a controlled manipulation of the message even without the ability to decrypt it.

This is probably the most dangerous attack on OSGP encrypted messages in the absence of strong authentication. Due to the simple message structure, the attacker can easily guess which part of the ciphertext corresponds to table entries which will be used to execute commands on the receiving device. Even if the attacker does not know which command will be executed instead it is possible to disturb the communication and possibly execute commands that disrupt communication at all.

Related-key attacks. If two messages are encrypted with the same key under a stream cipher, the XOR of the two ciphertexts equals the XOR of the two plaintexts. Thus, it is important to never use the same key twice. For this reason, modern stream ciphers use an initialization vector (IV) in addition to the key; the initialization vector can be publicly known, however they must be different for every encrypted message. The RC4 stream cipher does not specify an initialization vector. The common workaround is to use parts of the key as IV. This however is bad practice. An attacker who observes a certain number of IVs can reconstruct keys using correlated biases in the keystream. This was first observed by Roos [25] and formalized as attack by Fluhrer, Mantin, and Shamir [11], the famous 'FMS attack'. The common workaround to thwart the FMS attack is to drop the first 256 bytes of the keystream. Klein [18] however showed that dropping 256 bytes is not enough and that a much smaller number of observed IVs is sufficient to recover the key. This attack was put into practice and practically broke the WEP standard which sets the temporary key as `<init-vector|main key>`. This is similar to the way RC4 is used in the OSGP: the temporary key in OSGP is a random-looking 8-byte value (the digest of the request) xored to the first 8 bytes of the main key followed by the remaining 8 bytes of that key; see Figure 3. While this construction is slightly longer (in WEP the IV consists of 3 random bytes and a key of 5 bytes), the OSGP model is weaker given that the request digest is known. Furthermore, the OSGP allows a receiver to manipulate the sequence number, and thus have a control on how the initialization vector changes. Klein's *related-key attack* [18] on WEP reconstructs the key after observing only 13,000 sessions in WEP, i.e., different keystreams with different initialization vectors.

Attacking RC4 in the OSGP. Breaking the RC4 encryption in WEP can be carried out within seconds, as it is easy to observe a large number of sessions by means of packet spoofing. This is harder in the OSGP case, as there is no straightforward mechanism for an attacker to force the targeted system to generate encrypted messages.

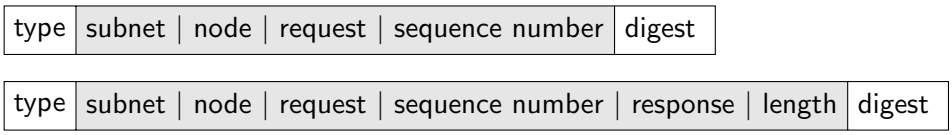

Figure 4: **Message format of OSGP requests and responses.** The digest is computed over the corresponding gray part and appended at the end.

However, in the OSG Protocol a receiver is supposed to reject a packet based on a wrong sequence number. The receiver not only rejects but also appends the correct sequence number to the NACK message. Thus, an attacker who can fake those NACKs is able to arbitrarily generate messages with chosen sequence numbers, which then can be used to derive the key. In particular, the attacker can try out related keys differing in a few well-chosen bits.

The way RC4 is used in the OSGP requires a new key setup for every request-response pair. It is not clear how big the performance advantage over a good AES implementation would be. Note that the common protection mechanism in the use of RC4 to drop the first 768 to 3,072 bytes of the keystream appears not to be done in the OSGP, and might impact efficiency.

More RC4 victims. Among the victims of recent attacks exploiting biases in the RC4 keystream were TLS cipher suites using RC4 for encryption and the HIVE hidden volume encryption system [2, 24]. It should be noted that there is no easy patch against the attack of AlFardan et al. [2] such as for the more implementation dependent attacks such as BEAST [7] and Lucky 13 [3], which are attacks against padding errors in CBC mode. It is now generally recommended to avoid usage and phase out RC4 where it is still used.

4 The OSGP digest function

This section introduces notation and a first brief analysis of the OSGP digest function and serves as basis for the attacks outlined in the following section.

Usage of the digest function. This function computes an authentication tag called *digest* on the (unencrypted) message. More specifically, as depicted in Figure 4 the digests are computed on the destination address (subnet address and node), the message bytes (payload), an appended sequence number, and in case it is response message also on the response and its the response length.

Linearity of the digest function. The OSGP digest function outputs 8 bytes of digest that are generated by processing the message byte by byte while going linearly through all key bits. In fact, the digest function turns the 96-bit OMA key into a 144-bit key: after processing all bits of the OMA key the function goes back to the first key bit and processes the first 48 bits once more. While this does not add entropy — or additional effort, as any attacker that has the last 96 bits immediately has the remaining 48 — it adds 48 additional rounds to the digest function.

Notation. For the ease of presentation we introduce the following notation. Where possible we try to stay close to the algorithm code given in Annex E of [10] computing the digest output in place.

i, j: Parameter i indexes *bytes*, parameter j indexes *bits*.

$k_i[j]$: This value denotes *bit* j of key byte i.

ℓ: We denote the number of *rounds* by ℓ. In total there are 144 rounds, exactly one per key bit.

m_ℓ: The value m_ℓ holds the message *byte* processed in round ℓ, also called the *active message byte*. Given that there are more rounds than message bytes m_ℓ is set to 0 once the algorithm has processed all message bytes.

$m_{i,j}$: When discussing the implementation it is useful to see the correspondence between the active key bit and the active message byte. Key bit $k_i[j]$ corresponds to the $(8 \cdot i + j)$th message byte which we also denote by $m_{i,j}$.

$S_{i,j}$: Internal state byte corresponding to key bit $k_i[j]$. The eight state bytes corresponding to the bits of the last key byte form the final digest value. In round 0, $S_{i-1,j}$ equals 0 for all j. When updating the state, the algorithm uses state $S_{i-1,j+1 \bmod 8}$, i.e., the state byte determined in round $\ell = (i-1) \cdot 8 + j + 1$ or equivalently 7 rounds previously; for $j = 7$ this is $S_{i,0}$.

$C_b(X)$: The function updates the active state byte. There are two possible outcomes depending on input bit b. Depending on key bit b, the result is plus or minus $\neg X$ shifted cyclically one bit to the left or to the right.

Updating the state. The implementation uses an array of 8 bytes for generating the digest. The algorithm cycles through those bytes and updates them one by one according to the currently active message and key bytes. For our analysis we use a linear description of the state to indicate how state byte $S_{i,j}$ is modified with input from key bit $k_i[j]$ and message byte $m_{i,j}$.

The code of the reference implementation in [10] in condensed write-up using j rather than $7 - j$, `k[i][j]` for key bit $k_i[j]$, `m[·]` for the active message byte, and `digestValueOut[j]` for the variable holding state byte $S_{i,j}$, is given as

```
n = ~(digestValueOut[j] + j );
if (k[i][j])
    digestValueOut[j] = ((n << 1) + (n >> 7));
else
    digestValueOut[j] = -((n >> 1) + (n << 7));
digestValueOut[j] += digestValueOut[ (j+1) % 8 ] + m[8*i+j];
```

In other words, the digest function adds the index j to the state $S_{i-1,j}$, negates the outcome, and stores the result in the temporary variable n. If key bit $k_i[j]$ is 1, shift the bits of n cyclically to the left and write the result to digest byte $S_{i,j}$. If key bit $k_i[j]$ is 0, shift the bits of n cyclically to the right and write the negated result to $S_{i,j}$. This is the only part depending on (a single bit of) the key. Finally, add state $S_{i-1,j+1 \bmod 8}$ and the active message byte to $S_{i,j}$.

Figure 5 displays the entire algorithm in our notation.

for $i = 0$ to 17
    for $j = 0$ to 7
        $S_{i,j} = C_{k_i[j]}(\neg(S_{i-1,j} + j)) + S_{i-1,j+1 \bmod 8} + m_{i,j}$

Figure 5: The OSGP digest function.

Differences between OSGP and EN 14908. The EN 14908 encryption function is essentially the same as the OSPG digest function, though has an additional parameter that defines the initial state. There is a small difference in the counting though, which does have a large impact: in OSPG, bits are counted backwards (the value of $7 - j$ is used, rather than j directly). This means that in EN 14908, the internal state added to the active state byte is the one that got updated in the previous round ($S_{i,j-1}$ in our notation). This means that a bit flip in $S_{i,j-1}$ affects the next 7 states. The counting change in the OSGP digest however yields that the internal state added to the active state byte is kept in the variable that gets overwritten in the next round ($S_{i-1,j+1 \bmod 8}$ as explained above). Thus bit flips are not further propagated in the OSGP digest.

Figure 6 displays the EN 14908 algorithm in our notation.

for $i = 0$ to 5
    for $j = 0$ to 7
        $S_{i,j} = C_{k_i[j]}(\neg(S_{i-1,j} + j)) + S_{i-1,j-1 \bmod 8} + m_{i,j}$

Figure 6: EN 14908 encryption function.

It does appear that EN 14908 was the original version, and OSPG got adapted to a different endianness in a way that slightly weakens it.

5 The meter as oracle: reconstruction of key and message

This section outlines how we can decrypt an encrypted message and reconstruct the OMA key used for authentication. We first present the general idea and then sketch how to recover the authentication key by showing explicitly how the last key byte can be derived from the last 9 rounds of the digest computation.

There are two ways how we can probe the system to reveal information about an encrypted message and the authentication key: using the receiving device as an oracle and using chosen messages.

Receiver as an Oracle. Upon receiving an encrypted and authenticated message the receiver will try to verify the digest. If the digest does not match an error message is returned. The device will return a response message including a negative-acknowledge character (NACK), that is a fixed error code of length 1, and a digest computed on the response message. In the OSG Protocol, this NACK message is explicitly never encrypted, turning the receiver into an oracle. Exploiting this, we make a controlled modification of the message and the digest, and use the receiver to test if the modification we made to the digest matches the modification we made to the message.

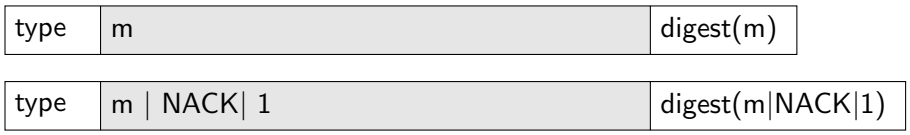

Figure 7: **Oracle attack.** On sending a request message with digest, the response of the meter contains the request, the one-byte error code, its length, and a digest computed on these partially known values.

Each such query will give us one bit of information about the key or the message; with $96 \cdot 2$ queries, we can reconstruct the entire authentication key. As the message encryption key is derived from the digest of the first message in a conversation, this message is not suitable for this attack; any modification of the digest would change the key, and thus got the message to decrypt wrongly (leading to a surely wrong digest). The response to the request however is independent of its own message digest, and thus can be used directly.

Note that the actual encryption key is derived from the digest, and will — after the modification — decrypt the message into complete nonsense. This does not affect our attack, as we only look at the digest value. It does mean, however, that the receiver will likely reject the messages even if the digest is correct, but then with a different error-code (e.g., wrong sequence number).

Chosen message attacks. If we can manipulate the sender to send a specific message, we can send two similar messages and compare the digests. This also gives us one bit of information per message pair, although a more elaborate attack is feasible where more bits are recovered at once. To some extent, message manipulation is easy. If the receiver sends the error message 'wrong sequence number', the sender will send the new message with the new sequence number, giving an attacker the chance to have the same message resend arbitrarily often, with the bytes representing the sequence number being of the attackers choice. Other attacks may be possible on an application level, if the protocol is used as a carrier for a more user defined protocol.

The last 9 rounds. To outline how to decrypt a message and to reconstruct the current OMA key bit by bit we analyze the final 9 rounds of the digest function. We first start analyzing a plaintext message with a digest and then extend the attack to an encrypted message.

Using the notation introduced in Section 4, the last 9 rounds of the protocol look as follows:

$$
\begin{aligned}
S_{16,7} &= C_{k_{16}[7]}(S_{15,7} + 7) + S_{16,0} + m_{16,7} \\
S_{17,0} &= C_{k_{17}[0]}(S_{16,0} + 0) + S_{16,1} + m_{17,0} \\
S_{17,1} &= C_{k_{17}[1]}(S_{16,1} + 1) + S_{16,2} + m_{17,1} \\
S_{17,2} &= C_{k_{17}[2]}(S_{16,2} + 2) + S_{16,3} + m_{17,2} \\
S_{17,3} &= C_{k_{17}[3]}(S_{16,3} + 3) + S_{16,4} + m_{17,3} \\
S_{17,4} &= C_{k_{17}[4]}(S_{16,4} + 4) + S_{16,5} + m_{17,4} \\
S_{17,5} &= C_{k_{17}[5]}(S_{16,5} + 5) + S_{16,6} + m_{17,5} \\
S_{17,6} &= C_{k_{17}[6]}(S_{16,6} + 6) + S_{16,7} + m_{17,6} \\
S_{17,7} &= C_{k_{17}[7]}(S_{16,7} + 7) + S_{17,0} + m_{17,7}
\end{aligned}
$$

To determine the last key bit $k_{17}[7]$, we modify a bit in message $m_{16,7}$. Suppose for now we know how to add the value '1' to this message. This means that $S_{16,7}$ can be increased by 1, as then is $S_{17,6}$. This yields two possibilities for $S_{17,7}$: in the shift function $C_{k_{17}[7]}$ the modified bit is either shifted to the right and added (add 128), or shifted to the left and subtracted (subtract 2). Thus, the correct value for the new $S_{17,7}$ is either the original value plus 128, or the original value minus 2. Using either a chosen message attack (let the sender authenticate 2 messages that differ in $m_{17,7}$), or using the meter as an oracle (adding 1 to $m_{17,7}$ and testing which of the two possible digests the receiver accepts), we can verify which choice was taken in $C_{k_{17}[7]}$. This way, we have now determined one key bit. With this key bit fixed, we can repeat the same method modifying $m_{16,6}$ and determining the choice of $CS_{17,6}$, thus successively determining all key bits of the authentication key.

When dealing with an encrypted message, we cannot be sure whether flipping the last bit adds or subtracts 1. This, however, only doubles the number of possibilities — the options are now +128, -128, +2 or -2. By testing which of the four options hold, we not only get the key bit, but implicitly also decrypt one message bit of $m_{11,7}$. The others can be decrypted the same way by adding/subtracting $2, 4, 8, \ldots$.

OSGP aspects in key recovery. There is a minor complication in that the message sent to the requester is not the same message that the digest is computed on. Rather, it is the 'response' part of the message, with other fields such as subnet, node, request, and sequence number. For simplicity, we assume we have a backward compatible device for which we do not need to include the response length. This way, the response sent back to the requester consists of exactly the last bytes of the message used for the digest.

For longer messages the first and last parts are digested with the same key bytes. In this case, we can directly reconstruct the key using above method. For shorter messages, the outlined method only allows us to recover some of the key bits. However, we do still profit from the fact that some key bits are used twice, as this means we do not need to go back to the beginning of the message to get the OMA key.

In summary, we should expect to be able to get most key bits with this method, and then get the remaining ones by different means (e.g., through a brute force attack). As the attack on the last 8 bytes, this attack only works on the response message, and cannot work directly on the request.

To fully recover the key with the outlined method we would need messages with 144 bytes. Otherwise, the receiver of a message will fill in the trailing zeros themselves, and thus not

allow the attacker to manipulate them. If such a message does not exist due to the length restrictions on request and response tables, this can be achieved by using a receiver with no message length byte, which will accept a message of every length[2]. Finally, it might well be feasible to overcome this with a more advanced cryptanalysis as in [16].

Differences between the OSGP and EN 14908. The oracle attack can also be applied to the EN 14908 encryption function. Though it is slightly harder given to the differences in counting that were explained in Section 4. However, the same attack patterns still work and the last 9 rounds are given as

$$
\begin{aligned}
S_{16,7} &= C_{k_{16}[7]}(S_{15,7} + 7) + S_{16,6} + m_{11,7} \\
S_{17,0} &= C_{k_{17}[0]}(S_{16,0} + 0) + S_{16,7} + m_{12,0} \\
S_{17,1} &= C_{k_{17}[1]}(S_{16,1} + 1) + S_{17,0} + m_{12,1} \\
S_{17,2} &= C_{k_{17}[2]}(S_{16,2} + 2) + S_{17,1} + m_{12,2} \\
S_{17,3} &= C_{k_{17}[3]}(S_{16,3} + 3) + S_{17,2} + m_{12,3} \\
S_{17,4} &= C_{k_{17}[4]}(S_{16,4} + 4) + S_{17,3} + m_{12,4} \\
S_{17,5} &= C_{k_{17}[5]}(S_{16,5} + 5) + S_{17,4} + m_{12,5} \\
S_{17,6} &= C_{k_{17}[6]}(S_{16,6} + 6) + S_{17,5} + m_{12,6} \\
S_{17,7} &= C_{k_{17}[7]}(S_{16,7} + 7) + S_{17,6} + m_{12,7}
\end{aligned}
$$

Simulating the attack on the OSGP function we increase the message byte $m_{16,7}$ by '1'. This increases not only $S_{16,7}$ but also the $S_{17,0}, \ldots, S_{17,6}$. The state $S_{18,7}$ is now affected in two ways: not only by the input for the shift function $C_{k_{17}[7]}$ but also by the increased value of $S_{17,6}$, thus increasing the attack cost for an encrypted message slightly though without real impact.

6 Meet-in-the-middle attack

If an attacker gets hold of one plaintext message and one corresponding digest, she can test all possible authentication keys to find the correct one (though it may be that several keys fit for a specific message, so in reality an attacker would probably need two message-digest pairs). In theory, given a 96-bit key, the effort for such a calculation should be 2^{96} computations.

However, the digest function is linear in the sense that every execution of the internal loop depends on exactly one key bit and the internal state. Thus, the active round is independent of all key bits that have not been processed up to that round. The attacker can easily compute in both directions. To this end, the attacker guesses the last bits of the key, and runs the algorithm backwards for those bits. She then creates a table with all the internal states for all combinations of trailing key bits. Then, she runs the algorithm forward with the leading key bits, and verifies if the resulting internal state is a value in the state table.

OSGP aspects in meet-in-the-middle attacks. This attack does need a reasonably advanced attacker, mostly because the outlined attacks require a comparatively high amount of resources. In terms of getting access to a message and the digest, it seems relatively

[2]It is not clear in the specification how exactly the message length byte is used, so it might be possible to do this for every receiver.

straightforward — the digest is communicated in the clear, and the communication is predictable enough that an attacker should be able to predict some of the plaintext messages (there is no randomness in any of the messages, so simple knowledge of the protocols will be sufficient here).

We note that many more optimizations and combinations with other attack methodologies can be applied here making attacks much more efficient. We refer to the independently developed techniques in [16].

Meet-in-the-middle attacks for the EN14908 encryption key. In the original EN 14908 protocol, the encryption function uses only a 48-bit key. As the "meet-in-the-middle" attack still works, this reduces the effective key length to 24 bit, which can be brute forced within seconds and requires only 4 megabytes of memory for the tables. It is also easy for an attacker to obtain a plaintext/ciphertext pair, as this is exactly what the challenge-response pair is. In OSPG, a 96-bit version of EN 14908 is suggested, which is still in reach of an advanced attacker, but does need sizable resources to break.

7 Conclusions

We identified a number of vulnerabilities in the OSG Protocol, some of which can be used to completely recover the authentication- and encryption keys. The outlined attacks are just examples how to exploit the inherent weaknesses of the digest function and the RC4 stream cipher.

The OSGP digest function does not adhere to any crypto standard, deviating from any best practice in the design of a secure message authentication code (MAC). The digest function is weak in the sense that each key bit affects only one message byte directly; this property allows the backward computation. In addition to replacing the digest function, adding some randomness to the message text may make it harder for an attacker to guess a plaintext message. Also RC4 should not be used (in this way). If a stream cipher should be used for smart-meter communications, a good choice would be the finalists of the eStream Contest [5]. If it is not an option to change ciphers, at least the recommendation to drop the first 768 bytes of keystream should be followed.

However, we believe that the only clean way to resolve the security issues in the OSGP is to completely replace some of the algorithms involved, especially the digest function.

Moreover, the standard does not seem to provide entity authentication that is assurance on the source of messages. Broadcasts are 'authenticated' using the weak digest function. This however provides only message authentication and not source authentication as would be done by appending a digital signature.

Lessons to take away. The main lesson must be that designers should set security and privacy goals for their system. Then a selection of cryptographic primitives should be made to achieve these goals. The absence of public-key cryptography to secure broadcasted firmware poses a very high risk. The encryption and message-authentication mechanisms were chosen to be lightweight, but not to be secure. While RC4 is probably friendlier to non-volatile memory it also needs a lengthy initialization to reduce the risk of biases in the key stream.

This leads directly to the second lesson: a standard such as the OSGP should use state-of-the-art cryptographic primitives. Rather than using a stream cipher for encrypting messages averaging 100 bytes a block cipher in authenticated operation mode should be used. Authenticated ciphers such as AES-CCM [8] or AES-GCM [22, 9] provide message authentication in

encrypt-then-authenticate fashion and not as done in the OSGP as encrypt-and-authenticate. The usage of an authenticated cipher would also simplify the problem in OSGP with setting up encryption and authentication keys.

If AES is not a feasible alternative due to performance or memory restrictions, an alternative would be a modern lightweight cipher such as PRESENT [4] which has been recently included in the ISO/IEC 29192 standard [15].

References

[1] Aircrack-ng. `http://www.aircrack-ng.org/` (accessed version December 17, 2014). (Cited on page 2).

[2] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS. In Samuel T. King, editor, *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 305–320. USENIX Association, 2013. (Cited on pages 6 and 8).

[3] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 526–540. IEEE Computer Society, 2013. (Cited on page 8).

[4] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsø. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007. (Cited on page 15).

[5] Carlos Cid and Matt Robshaw. The eSTREAM Portfolio in 2012, 2012. `http://www.ecrypt.eu.org/stream/` (accessed version December 17, 2014). (Cited on page 14).

[6] Cisco. Next Generation Encryption, 2014. Published April 2012, last updated April 2014. `http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html`. (Cited on page 6).

[7] Thai Duong and Juliano Rizzo. Here come the XOR Ninjas, 2011. Unpublished manuscript. (Cited on page 8).

[8] Morris J. Dworkin. SP 800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. Technical report, Gaithersburg, MD, United States, 2004. (Cited on page 14).

[9] Morris J. Dworkin. SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, Gaithersburg, MD, United States, 2007. (Cited on page 14).

[10] European Telecommunications Standards Institute (ETSI). Group specification GS OSG 001: Open Smart Grid Protocol, January 2012. (Cited on pages 1, 3, 8, and 9).

[11] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001. (Cited on pages 6 and 7).

[12] Scott R. Fluhrer and David A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2000. (Cited on page 6).

[13] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (Non-) Random Sequences from (Non-)Random Permutations – Analysis of RC4 Stream Cipher. *J. Cryptology*, 27(1):67–108, 2014. (Cited on page 6).

[14] International Organization for Standardization. ISO/IEC 14908-1:2012:Information technology – Control network protocol – Part 1: Protocol stack, 2012. (Cited on pages 1, 3, and 4).

[15] International Organization for Standardization. ISO/IEC 29192-2:2012: Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers, 2012. (Cited on page 15).

[16] Philipp Jovanovich and Samuel Neves. Practical Cryptanalysis of the Open Smart Grid Protocol. 2015. To appear in the proceedings of FSE 2015. Copy received on January 27, 2015; prior to publication in a private communication. (Cited on pages 3, 13, and 14).

[17] Kali Linux. `https://www.kali.org/` (accessed version December 17, 2014). (Cited on page 2).

[18] Andreas Klein. Attacks on the RC4 stream cipher. *Des. Codes Cryptography*, 48(3):269–286, 2008. (Cited on pages 2 and 7).

[19] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001. (Cited on page 5).

[20] Subhamoy Maitra, Goutam Paul, and Sourav Sengupta. Attack on broadcast RC4 revisited. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2011. (Cited on page 6).

[21] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2001. (Cited on page 6).

[22] David A. McGrew and John Viega. The Galois/Counter Mode of operation (GCM), 2004. Submission to NIST Modes of Operation process. (Cited on page 14).

[23] Microsoft. Microsoft Security Advisory 2868725, 2013. Published 12 November 2013. `https://technet.microsoft.com/en-us/library/security/2868725.aspx` and `http://blogs.technet.com/b/srd/archive/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4.aspx`. (Cited on page 6).

[24] Kenneth G. Paterson and Mario Strefler. A Practical Attack Against the HIVE Hidden Volume Encryption System. 2015. To appear in the proceedings of ASIACCS'15. `http://eprint.iacr.org/2014/901/`. (Cited on pages 6 and 8).

[25] Andrew Roos. A Class of Weak Keys in the RC4 Stream Cipher, 1995. Two posts in sci.crypt, message-id `43u1eh$1j3@hermes.is.co.za` and `44ebge$llf@hermes.is.co.za`. (Cited on pages 6 and 7).

[26] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Discovery and exploitation of new biases in RC4. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2010. (Cited on page 6).

[27] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Statistical Attack on RC4 - Distinguishing WPA. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 343–363. Springer, 2011. (Cited on page 6).

[28] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2007. (Cited on pages 2 and 6).

[29] Serge Vaudenay and Martin Vuagnoux. Passive-only key recovery attacks on RC4. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 2007. (Cited on page 6).