V2G Injector: Whispering to cars and charging units through the Power-Line

Sébastien Dudek, Jean-Christophe Delaunay and Vincent Fargues
sebastien.dudek@synacktiv.com
jean-christophe.delaunay@synacktiv.com
vincent.fargues@synacktiv.com

Synacktiv

Abstract. Since vehicles became connected to a bus called CAN (Controller Area Network), many "garage" hackers got interested in investigating the different controllers, known as ECUs (Engine Control Units), and accessible via the On-Board Diagnostics (OBD) port. Among those different controllers, some of them are accessible via Wi-Fi, others via GPRS, 3G and 4G mobile networks, that could be attacked during a radio interception attack [19]. Moreover, another little-known vector of attack will appear with the deployment of V2G (Vehicle-to-Grid) systems that communicate via power lines support. Nevertheless, no public tool exists to interface with these systems, but also to analyse and to inject V2G traffic. That is why we have developed a tool called *V2G Injector* to attack these systems.

In this article, we will briefly introduce the V2G concept and its similarities with domestic Power-Line Communication systems. Then, we will present the techniques we use in our tool that aim to interface with the system, monitor and inject traffic. We will also present a new specification vulnerability in the communication medium we have been able to exploit to intrude the V2G network. To finish, we will talk about issues we have found during our tests on real equipment, and mitigations we can encounter, or apply, in some contexts as well as possible bypasses.

1 Introduction: rise of V2G

Due to its environmental friendliness, Electric Vehicle (EV) is gaining popularity in U.S.A, Japan, China and some countries in Europe. For example, by 2020, France aims to sell 2 million EVs, China 5 million, and Germany 1 million. As a consequence, global EV battery capacity keeps increasing.

Meanwhile, solar and wind energy output are variable and difficult to predict accurately, so their production cannot follow consumer demand patterns. Their price variability has therefore increased during the day, which strengthened the business case for energy storage.

Therefore, energy storage systems have been developed:
— Battery-to-Grid (B2G), which stores energy in dedicated batteries;
— Vehicle-to-Grid (V2G), which uses Electric Vehicles (EV) to store energy. Car owners are also remunerated when plugging into a bidirectional charging/discharging system, mostly to compensate possible deterioration of the battery.

A simplified V2G architecture is shown in figure 1.

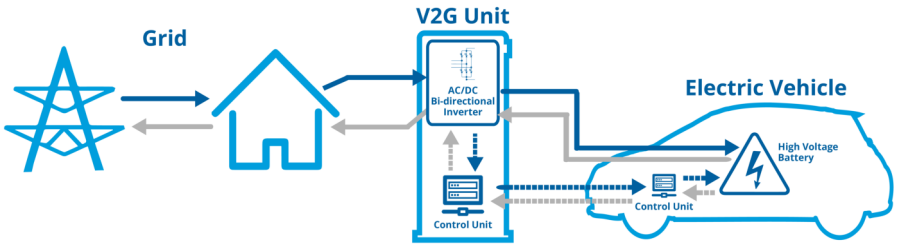

Fig. 1. V2G architecture (source: [1]).

But without interoperability between EV, charging station also known as EV Supply Equipment (EVSE), and backends, these technologies could not be practical and be difficult to sell. To address this issue, people in the industry have designed standards such as:
— ISO/IEC 15118 [9, 15, 16]: Vehicle-to-Grid (V2G) communication;
— IEC 61851 [13, 14]: conductive charging systems;
— IEC 61850-90-8 [11]: communication networks for EVs;
— Technical specifications from CHAdeMO (CHArge de MOve) [8];
— Documents from DIN (Deutsches Institut für Normung; in english, German Institute for Standardization) [10].

The physical connection and protocols between an EV and an EVSE are described in the following sections.

2 V2G communication

2.1 V2G ECU

A typical V2G vehicle ECU, more precisely a Vehicle Charge Control Unit (VCCU), is shown in figure 2. It is interfaced with a Combined Charging System (CCS), that has a connector format depending on the current country standard. Moreover, this ECU can also be directly connected to the CAN bus of the vehicle, or through a gateway.

The ECU is responsible for the following tasks:
— coordination of charging-related vehicle functions and HV-switches between inlet and DC-link;
— vehicle state management;
— communication with the backend;
— etc.

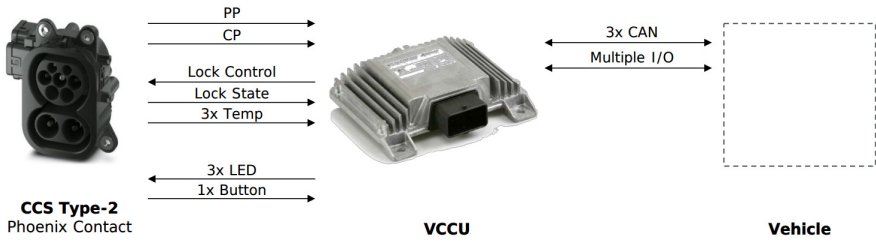

Fig. 2. V2G ECU (source: [20]).

As shown in figure 3, the ECU is composed of two main chips:
— a host CPU chip (generally specific to the automobile area);
— a QCA modem used to communicate in PLC using HomePlug AV/GP standard.

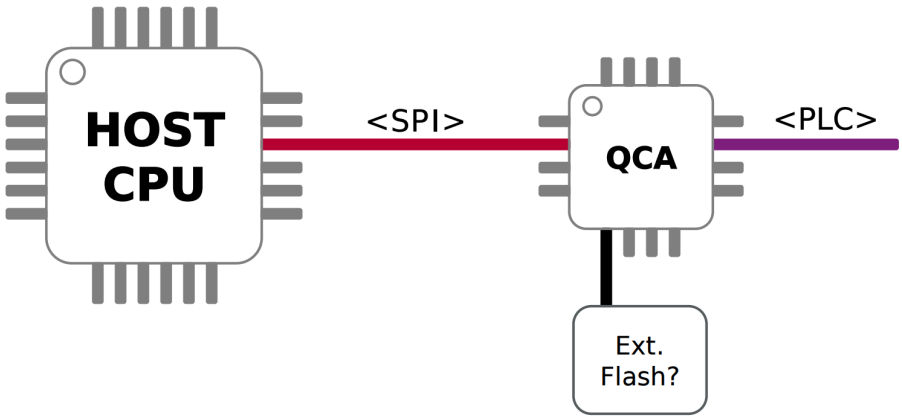

Fig. 3. ECU components (source: [20]).

2.2 Architecture

When a vehicle is plugged to a charging station, a communication is established between two controllers:

— the EV Communication Controller (EVCC) that acts as a client;
— and a Supply Equipment Communication Controller (SECC) that is the server part.

As we can see in figure 4, the communication is performed using two standards: IEC 61851 for low-level, and ISO/IEC 15118 for high-level communications. These two layers are incompatible, but the IEC 61850-90-8 standard [12] indicates how to bridge them.

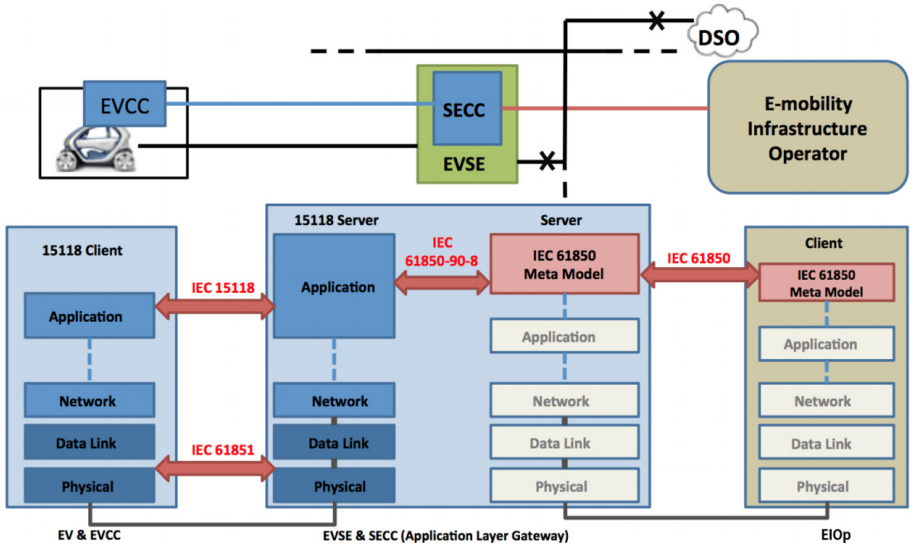

Fig. 4. V2G network architecture (source: [22]).

IEC 61851 is used for basic signalling, to exchange information about the voltage level and Pulse Width Modulation (PWM) through the control pilot line we will see in later sections. More complex information is left to the higher level ISO/IEC 15118 (V2G layer).

2.3 V2G layer

Figure 5 illustrates the OSI layers defined by ISO/IEC 15118. When EV and EVSE communicate through V2G, the controllers do it by bridging ISO/IEC 15118 messages to IEC 61850. V2G messages are exchanged over

IPv6. After plugging the charging cable, the SECC Discovery Protocol (SDP) on port UDP 15118 is used to forward the EV traffic to the right IPv6 address and port of the appropriate V2G server to reach. This protocol handles the security mode requested by the EVCC and the answer normally acknowledges that. Then all data in V2G is transported at V2GTP layer.

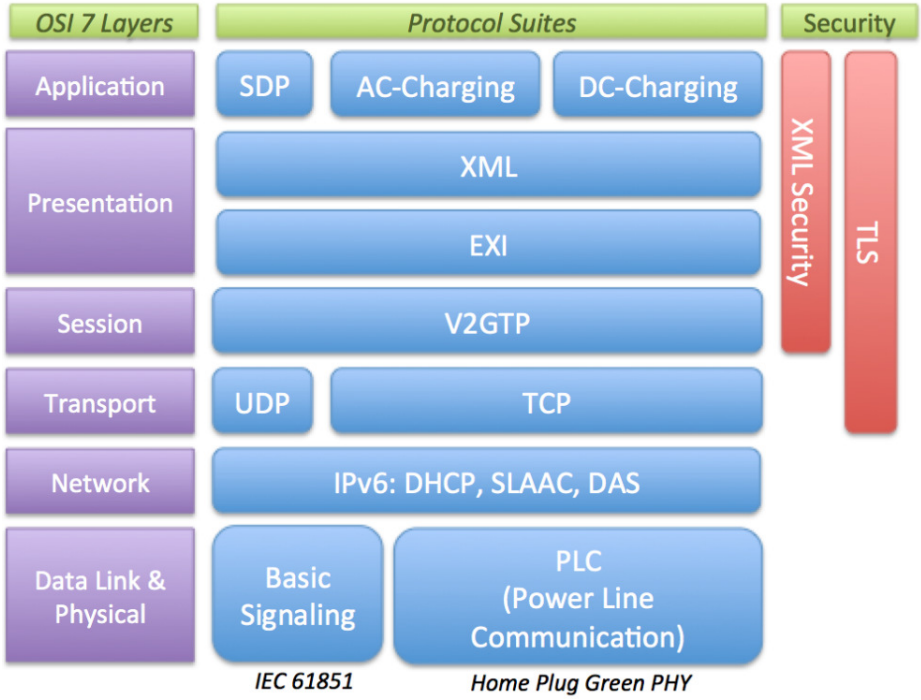

Fig. 5. V2G protocol stack (source: [22]).

Data exchanged in V2GTP layer are generally XML files and are encoded/decoded with the EXI compression algorithms, depending on their XSD format definition. The ISO/IEC 15118 also allows to specify a set of symmetric and asymmetric cryptographic algorithms.

Noteworthy characteristics of layers SDP, V2GTP and EXI are described in the following sections.

2.4 Secure communications

When enabled, TLS session information and certificates are negotiated after the TCP connection is established. Sensitive data are then encrypted, and are contained in signed XML stream (see figure 6).

To use the secure communication, an EV must have two distinct private keys and certificates (Contract and OEM Prov, see figure 7) to ensure encryption and authenticity at the same time. The EVSE has one private key and certificate to establish the TLS communication. This feature generally prevents interception of V2GTP data, but to be able to check the authenticity of the SECC public key, a Certificate Authority (CA) is required.

Nevertheless, it should be reminded that V2G EVCC should work in heterogeneous systems (domestic units, dedicated power stations, and so on). So it is not surprising to see permissive V2G implementations and configurations in the wild. This will lead us to questions when testing this type of environment, for example:

— is there a control of the security mode requested by the SDP protocol?
— how EV and EVSE certificate checks are really implemented?
— how the XML signature is really checked?

HomePlug GreenPHY (HPGP) Power-Line Communication (PLC) has been adopted by the V2G standard for the physical communication medium (see figure 5), and this medium also includes security mechanisms to encrypt data exchanged in the Power-Line.

3 HomePlug GreenPHY

3.1 HomePlug AV and GreenPHY

Car connected to charging stations use the HomePlug GreenPHY [4] (HPGP) specification, that is in fact a subset of the HomePlug AV [3]. The HomePlug GreenPHY is intended to be used in the “smart” grid, to plug Electric Vehicles on V2G units and are fully interroperable with the AV specification. As shown in figure 8, the HPGP has decreased throughput as it exclusively uses Quadrature Phase Shift Keying (QPSK) instead of very high orders of Quadrature Amplitude Modulation (QAM). So the peak PHY rate is the main difference we can observe when interconnecting HomePlug AV and GP together.

As in the AV specification, HPGP has two kinds of keys to manage and encrypt data on the Power-Line:

Fig. 6. Signed V2G message (source: [7]).

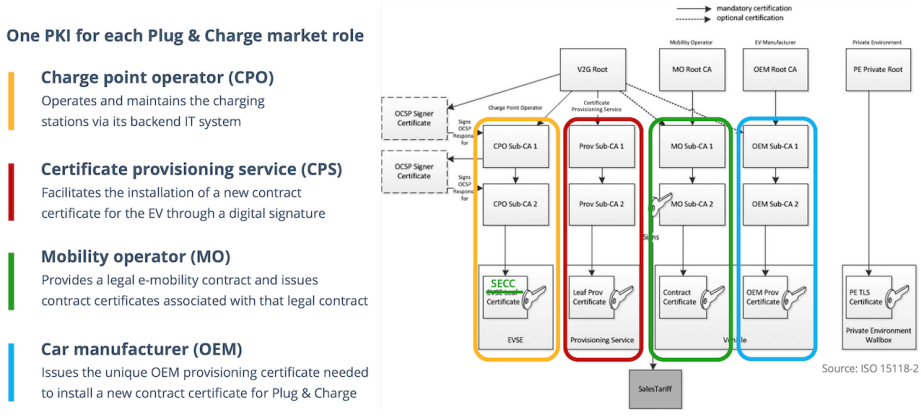

Fig. 7. PKIs as defined by ISO 15118 (source: [7]).

— Network Membership Key (NMK): to encrypt the communication using 128-bit AES CBC;
— Direct Access Key (DAK): to remotely configure the NMK of a targeted PLC device over the Power-Line interface.

HomePlug GP PHY Simplifications Reduce Cost & Power Consumption

	Parameter	HomePlug AV	HomePlug GP
PHY	**Spectrum**	2 MHz to 30 MHz	2 MHz to 30 MHz
	Modulation	OFDM	OFDM
	# Subcarriers	1155	1155
	Subcarrier spacing	24.414 kHz	24.414 kHz
	Supported subcarrier modulation formats	BPSK, QPSK, 16 QAM, 64 QAM, 256 QAM, 1024 QAM	QPSK only
	Data FEC	**Turbo code** Rate ½ or Rate 16/21 (punctured)	**Turbo code** Rate ½ only
	Supported data rates	**ROBO:** 4 Mbps to 10 Mbps **Adaptive Bit Loading:** 20 Mbps to 200 Mbps	**ROBO:** 4 Mbps to 10 Mbps

Fig. 8. Simple of HomePlug GP and AV comparison (source: HomePlug GreenPHY 1.1 white paper [4]).

Moreover, HomePlug AV defines a pairing button mechanism that allows to easily setup the NMK of PLC devices and join an AV Logical Network (AVLN) [23]. Alternatively, HomePlug GP has important mechanism used for Plug-in Electrical Vehicle (PEV) association that do not require any actions (button, or NMK configuration with the DAK) when an EV is plugged on an EVSE.

3.2 Plug-in Electrical Vehicle (PEV) Association

As described in the HomePlug GreenPHY white paper, PEVs may be charged at home, work and in public areas. But at the beginning, a HomePlug GP EVCC is unconfigured and needs to join the AVLN of the EVSE when charging cable is plugged in. Unfortunately, things are complex in Power-Line, because all PLC packets are broadcasted through in the Power-Line, so a PEV can be seen by multiple EVSEs and vice versa as shown in figure 9. So consumer billing, utilities and auto manufacturers could be concerned by a number of security-related matters if their PEV connects itself to a wrong charging station.

To avoid security issues like bad associations and billing errors, HPGP defines a “security mechanism” called SLAC (Signal Level Attenuation

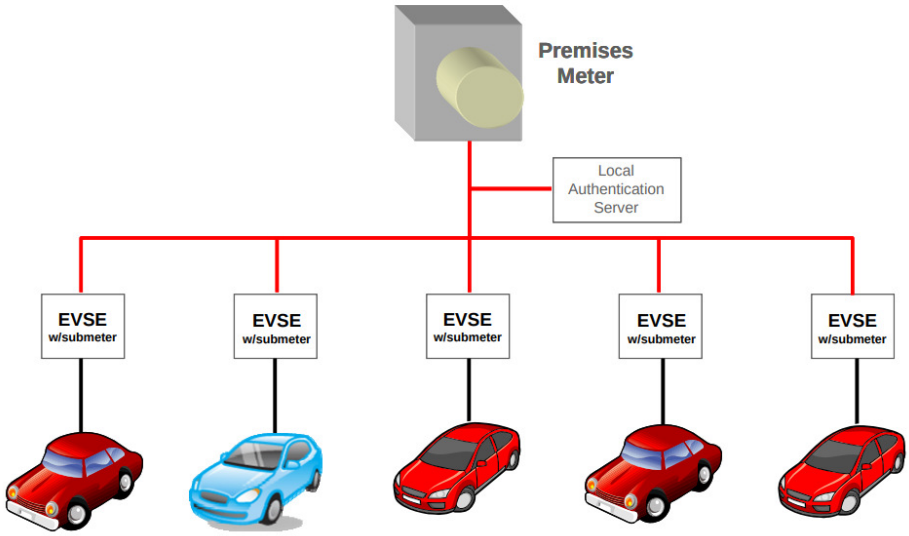

Fig. 9. SLAC Procedure Facilitates PEV/EVSE Association (source: HomePlug GP white paper).

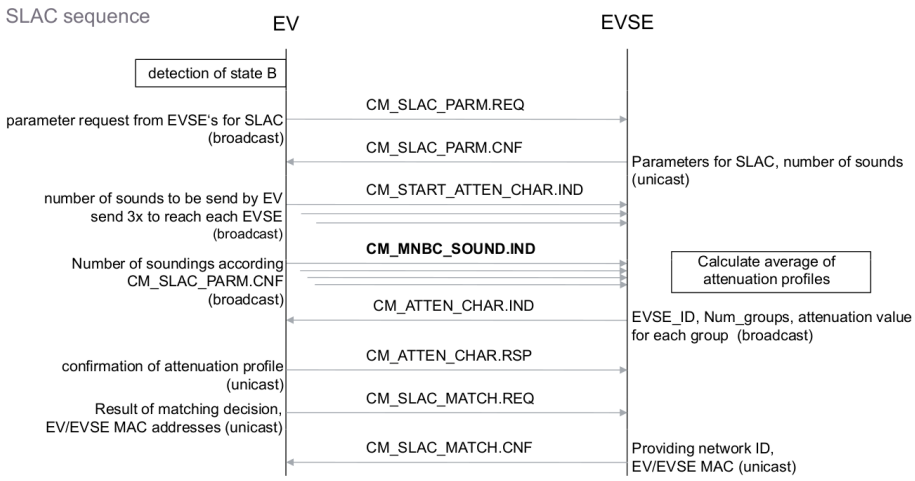

Fig. 10. SLAC Sequence (source: [21]).

Characterization). The principle of this mechanism is to have the PEV broadcast a series of short unacknowledged SOUNDING packets, so that stations in the range can measure a received power and send it to the client that will select the MAC address which has the highest received signal. The sequence is illustrated in figure 10.

From an attacker's point of view who emulates a fake charging station, this mechanism is not blocking and can be bypassed by transmitting tampered attenuation values, to force targeted PEVs to connect to the AVLN of our fake EVSE. However, we will see in the next sections that the SLAC mechanism has also weaker points that allow an attacker to steal the NMK key distributed by a legit EVSE.

4 State of the Art

4.1 Publications

Many articles have been published about the use and impacts of these technologies, but only few of them tackle the security of such systems, by discussing some security aspects, privacy issues and possible improvements to make on V2G [26, 27]. However, most of the stated attacks are possible but have not yet been demonstrated in practice.

Moreover, V2G systems communicate through the Power-Line as a physical support. Indeed, as we will see in further sections, the HPGP standard is used to transport data. Some publications regarding HomePlug AV and its weaknesses have already been published [18] and some of the enumerated attacks could be used against V2G units and PEVs PLCs in certain cases, as HGPG is fully interroperable with the AV standard.

4.2 Existing tools

We observed that the V2G is rather a closed world:
— official specifications are not free;
— the rare available tools for analysis are expensive and hard to get;
— the device to interface with is not easily accessible;
— and no arbitrary frame injector equipment actually exists.

But thankfully we can find documentation, and the very useful open source implementations *RISE-V2G* [25] and *OpenV2G* [24], that allow us to simulate V2G communications between an Electric Vehicle and a charging station.

Analysis software exist such as the *V2G Viewer pro* of HSE Electronic which is not free. However, this software has a demo version that limits

analysis to 100 network packets. So we used this software to help us understand all layers from our captures at the beginning.

We also observed that HomePlug GreenPHY dissectors for Wireshark were missing. But V2G Viewer pro demo in combination with HomePlug GreenPHY specifications helped us understand and implement HomePlug GreenPHY packets at MAC layer, and layers relative to SECC and V2GTP a bit faster.

4.3 Our contribution

To connect to a V2G network, we use a development kit for HomePlug GreenPHY we will introduce later in this article. We have implemented missing Scapy layers to decode/encode HPGP, SECC and V2GTP packet. Moreover, we have updated the HomePlugPWN [17] tool to support attacking V2G implementations based on HomePlug GreenPHY and have written an EXI data encoder/decoder based on the *RISE-V2G* shared Java library to analyse and inject V2G frames.

During our tests, we also found a flaw in the SLAC procedure of the HPGP standard. To reduce the costs of buying a development kit, we are currently working on an adaptation to use a domestic adapter instead.

5 Intruding a V2G network

5.1 Data propagation over Combined Charging System connectors

To intrude the network, we need to interface somehow. Indeed, to plug-in with a car, or a charging station, we can find many different types of Combined Charging System connectors. Within Europe, the IEC 62196 [5] Type 2 connector is largely used, and its pinout (see figure 11) is as follows:

— PP: Proximity pilot for pre-insertion signalling;
— CP: Control Pilot for post-insertion signalling;
— PE: Protective earth;
— N: Neutral (single/3 phase AC/DC-mid);
— L1, L2 and L3 three phase AC/DC-mid.

To interact with V2G systems, it should be noted that HomePlug GreenPHY data are multiplexed onto the Control Pilot and ground lines. But, interfacing with a male or female connector is not the only way to start intruding the network.

5.2 Data Propagation over Power-Line

It is important to note that data over Power-Line is superposed on the power supply [18], so that information can propagate through many installations depending on signal strength. So if a charging station shares, at least, the same column heading as another building, or any other domestic installation, then the PLC of a resident may be able to see and communicate with the charging stations' PLC modem.

Previous work has also shown that theoretically choke-coils can be used in new installations to attenuate high frequency signal propagating through the Power-Line. Unfortunately, these choke-coils are not installed everywhere, and if there are used on new installations, in practice these components do not ensure a long-term effect and could be less precise due to wear [2, 17].

5.3 Required hardware

To interface with the V2G network, we acquired a Devolo development kit for approximately 200€, as pictured in figure 12, that exposes three interesting PLC interfaces:

— on top-middle a Power-Line Communication module based on the QCA7000 (QCA7k) by Qualcomm Atheros;
— on top-right a twisted pair line interface, as well as a coax SMA female interface.
— and at the bottom an AC coupler interface to plug-in with a domestic plug.

For our case, twisted pair interfaces can directly be used to connect them to an IEC 62196 connector (we some adaptation). But as mentioned earlier, the AC coupler maybe interesting to use to test if the system shares the same electrical network as the plug we are connected to.

The kit also has a privileged Ethernet interface (local interface) that could be used to set-up the PLC modem without needing to use the Direct Access Interface (DAK).

5.4 HomePlug keys

As it is a subset of the AV specification, HPGP uses the same 2 types of keys as Homeplug AV:

— NMK: Network Membership Key used to create or/and join a HomePlug network;
— DAK: Direct Access Key.

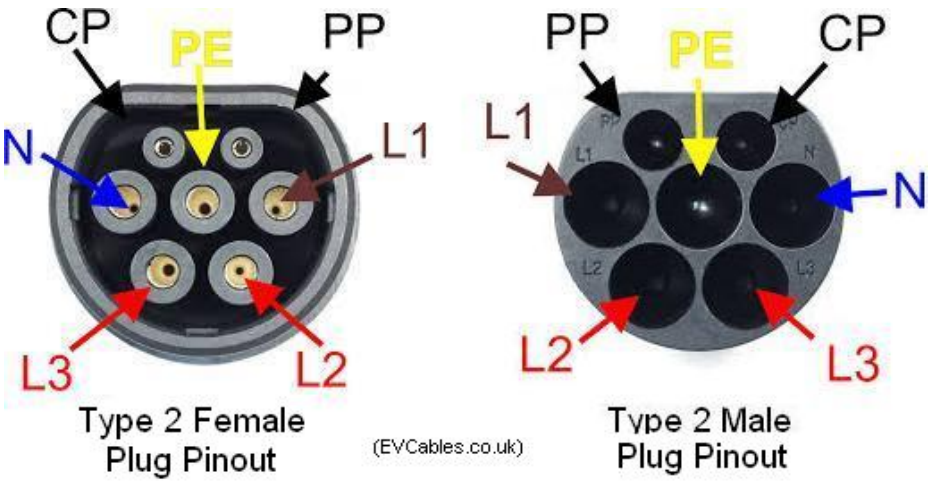

Fig. 11. IEC 62196 Type 2 pinout.

Fig. 12. Devolo HomePlug GP devkit.

The NMK can be set directly using a local interface by sending a *Set Encryption Key Request* HomePlug AV packet. On the other side, if users are not connected to the local interface of the PLC device, they can set the NMK remotely via Power-Line by using the correct DAK key associated to the remote device.

Previous works on HomePlug AV devices [18], have shown that the DAK key was generated by deriving the MAC address of the PLC device with a known algorithm, so it was possible to quickly find the DAK of all Central Coordinators PLC devices. Those weaknesses could also be observed if vendors of V2G PLCs use a similar previsible technique to generate the DAK.

5.5 Detection of HomePlug AV/GP devices

The HomePlugPWN tool suite provides a script called *plcmon.py* that enables the "sniff mode" feature on the PLC (see figure 13) and detects the presence of PLCs that behave as Central Coordinators (CCo) in the Power-Line support. In domestic networks, a CCo the PLC that is generally connected to the internet router. In our context, the CCo is always the EVSE.

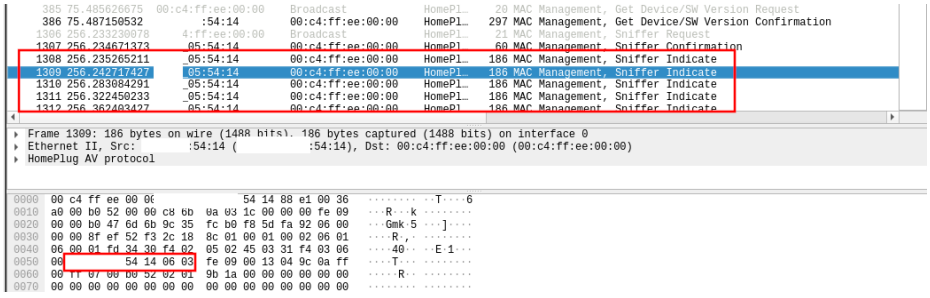

Fig. 13. Wireshark capture with Sniff indicates packets received when enabling sniff mode on the local PLC.

By using this feature, an attacker can detect charging stations if they are on the same electrical network and if the signal is strong enough.

5.6 HomePlug GreenPHY modes

Before going further, it should be noted that HPGP modems can be configured in three specific modes: unconfigured (to act like a domestic plug), PEV, or EVSE.

It is important to be aware of these different modes, because abstracted packets to the local interface may differ from one mode to another, especially when packets are specific to HPGP.

5.7 A design flaw in the SLAC procedure

By analysing the SLAC sequence, as observed in figure 10 from the specifications, the first attack that comes to mind is to craft a precise *CM_ATTEN_CHAR.IND* response to force the PEV to connect to our fake charging station. On the other side, a fake PEV can always try to start a SLAC sequence by connecting some charging stations.

But after implementing Scapy layers for HPGP, and switching the PLC modem to PEV mode, it was found that any PEV is able to sniff the NMK sent by all EVSE during SLAC processes in clear. Indeed, Management Message Entry (MME) packets, like those used for the SLAC procedure, are broadcasted over the Power-Line and are generally not encrypted, so anyone connected in the same electrical network can capture SLAC procedure related messages. To observe these packets, we have to change the mode of our PLC to PEV.

To act as a PEV, the modem's PIB has been dumped with *pibdump*, then the byte `0x1653` associated SLAC mode as to be modified with *setpib* tool, from *open-plc-utils* [6], as follows:

```
$ pibdump PIBdump.pib
[...]
$ setpib PIBdump.pib 1653 byte 1
```

Listing 1. Changing SLAC mode.

Then during a SLAC procedure, we can sniff incoming packets from the LAN interface of our PLC kit and observe the following HPGP packets corresponding to packets sent by the EVSE:

```
###[ Ethernet ]###
  dst       = bc:f2:af:f1:00:03
  src       = 00:01:85:13:43:11
  type      = 0x88e1
###[ HomePlugAV  ]###
     version   = 1.1
     HPtype    = 24677
     Reserved  = 0x0
###[ CM_SLAC_PARM_CNF ]###
        MSoundTargetMAC= ff:ff:ff:ff:ff:ff
        NumberMSounds= 10
        TimeOut   = 6
        ResponseType= 1
        ForwardingSTA= bc:f2:af:f1:00:03
```

```
        ApplicationType= 0
        SecurityType= 0
        RunID      = '+\x43\xee\xda\xff\x05\xa7\x34'
[...]

###[ CM_ATTEN_CHAR_IND ]###
        ApplicationType= 0
        SecurityType= 0
        SourceAdress= bc:f2:af:f1:00:03
        RunID      = '+\x43\xee\xda\xff\x05\xa7\x34'
        SourceID   = ''
        ResponseID= ''
        NumberOfSounds= 10
        NumberOfGroups= 58
        \Groups     \
[...]
```

Listing 2. Changing SLAC mode.

The last packet sent from an EVSE during a SLAC procedure is the *CM_SLAC_MATCH.CNF*, as shown in figure 14. We were able to decode the variable field, as shown in figure 15, that contains the NMK to join the new private network negotiated between the PEV and EVSE.

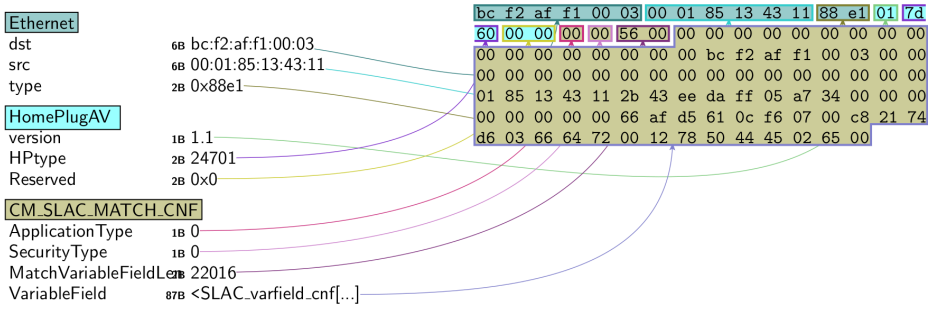

Fig. 14. *CM_SLAC_MATCH.CNF* message from EVSE.

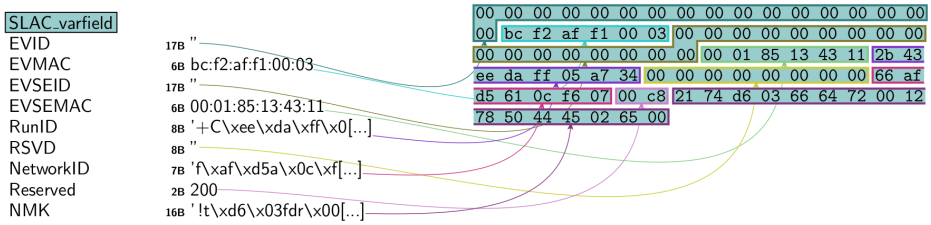

Fig. 15. *CM_SLAC_MATCH.CNF* message decoded.

We are then able to setup our PLC kit to join a targeted network by sending a *CM_SET_KEY.REQ* or by simply configuring *open-plc-utils-master/slac/pev.ini* with captured NMK and Network ID and running *pev* tool.

5.8 Into the AVLN

Once a device is part of an AVLN (AV Logical Network), it is able to talk to every possible service on the network, depending on access controls. The device is also able to perform a network discovery to see PLCs on the same AVLN.

At this stage, an attacker in the same AVLN can try to discover available services on the EVCC and SECC parts. Generally, there is nothing interesting in the EVCC part, as it acts as a client that only sends data to the EVSE. On the other side, the SECC part is very interesting as it can expose many services such as SSH, maybe web managing servers, and many others depending on the constructor. But in this article, we will only focus on way we can talk over V2G.

By default, attacker's PLC kit can be set in promiscuous mode to intercept all packets, but two kinds of Man-In-The-Middle attacks can be performed:

— the classical way with an ICMPv6 Neighbour spoofing attack;
— or by racing the SECC procedure.

5.9 Racing the SECC procedure

This attack is optional in case we want to inject traffic, and could help to be more stable than an ICMPv6 Neighbour spoofing attack. But during this procedure, the attacker has to be fast by retrieving the NMK, configuring his PLC kit and then sending fake SECC answers for a while.

Indeed, since we are able to decode SECC layers with Scapy, we are able to capture a first message that is multicasted by the EVCC as follows:

```
###[ Ethernet ]###
  dst       = 33:33:00:00:00:01
  src       = bc:f2:af:f1:00:03
  type      = 0x86dd
###[ IPv6 ]###
     version   = 6
     tc        = 0
     fl        = 0
     plen      = 18
     nh        = UDP
     hlim      = 64
```

```
     src       = fe80::bef2:afff:fef1:3
     dst       = ff02::1
###[ UDP ]###
        sport     = 60806
        dport     = 15118
        len       = 18
        chksum    = 0xc9c7
###[ SECC ]###
           Version   = 1
           Inversion = 254
           SECCType  = SECC_RequestMessage
           PayloadLen= 2
###[ SECC_RequestMessage ]###
              SecurityProtocol= 16
              TransportProtocol= 0
```

Listing 3. SECC Request.

Then the SECC server listening on UDP port 15118 should send the following answer:

```
###[ Ethernet ]###
  dst       = bc:f2:af:f1:00:03
  src       = 00:01:85:13:43:11
  type      = 0x86dd
###[ IPv6 ]###
     version   = 6
     tc        = 0
     fl        = 278181
     plen      = 36
     nh        = UDP
     hlim      = 64
     src       = fe80::201:85ff:fe13:4311
     dst       = fe80::bef2:afff:fef1:3
###[ UDP ]###
        sport     = 15118
        dport     = 60806
        len       = 36
        chksum    = 0x3756
###[ SECC ]###
           Version   = 1
           Inversion = 254
           SECCType  = SECC_ResponseMessage
           PayloadLen= 20
###[ SECC_ResponseMessage ]###
              TargetAddress= fe80::201:85ff:fe13:4311
              TargetPort= 56330
              SecurityProtocol= 16
              TransportProtocol= 0
```

Listing 4. SECC Response.

As we can see in listing 4, to perform a Man-In-The-Middle attack, an attacker can try to send a crafted SECC answer with an arbitrary IPv6 address and port to join a fake SECC server. Moreover, another

interesting field to craft is the *SecurityProtocol* one that confirms the demanded security level (clear-text, or TLS), which could potentially downgrade V2G communication security by forcing the PEV to talk in clear-text.

During our research project, we did not find any system using the TLS feature yet. But to study the protocol more thoroughly, we have also tested the opensource solution *RISE-V2G* against *SecurityProtocol* tampering, but the EVCC part of this solution seems to check this field and stopped the communication if required security level is different from server's response. But implementation could vary between different manufacturers, so it could be interesting to look at this procedure in real life when a vendor uses it.

5.10 Analysing V2G packets

Once we are connected to a targeted AVLN and are able to intercept packets between a charging station and a car, we may see many mysterious IPv6 packets as shown in figure 16.

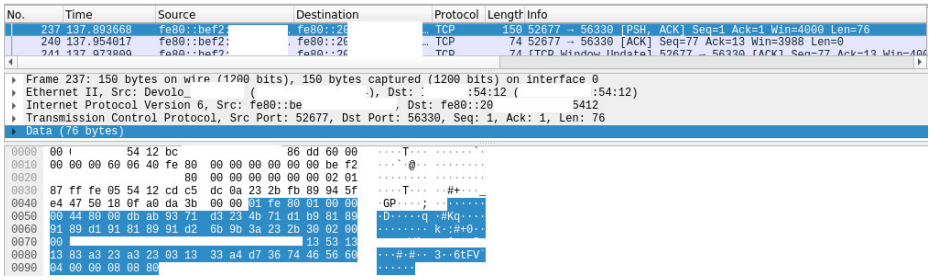

Fig. 16. Captured data exchange between a PEV and a EVSE.

According to IEC/ISO 15118, data are exchanged in an XML encoded format called EXI (Efficient XML Interchange). The encoding is documented and implemented [1] in C/C++, Java, and JavaScript. Moreover, this encoding supports many Web formats:

— XML (and formats using XML syntax, e.g., SVG, RSS, MathML, GraphML...);
— HTML;
— JSON (Java, JavaScript, C);
— CSS (EXI overview presentation for CSS);

1. `https://exificient.github.io/`

— JavaScript.

To compress data as much as possible, a specific grammar must be provided to the EXI encoder. V2G uses its own grammar that can be found in *RISE-V2G* project [2] in following XML Schema Definition schemas:

— `V2G_CI_AppProtocol.xsd`;
— `V2G_CI_MsgDef.xsd`;
— `V2G_CI_MsgHeader.xsd`;
— `V2G_CI_MsgBody.xsd`;
— `V2G_CI_MsgDataTypes.xsd`;
— `xldsig-core-schema.xsd`.

Precisely, each V2G type of message uses its own grammar as follows:

— AppProtocol → `V2G_CI_AppProtocol.xsd`;
— XMLSIG → `xldsig-core-schema.xsd`;
— and MSG most of the time → `V2G_CI_MsgDef.xsd`.

To encode/decode packets, we use the *RISE-V2G* shared library embedding the EXIficient framework in Java. But as EXI data do not store the current context, we have created a naive data type iterator that tries possible solutions to decode current packet without its context information. An example of the iterator is shown in listing 5.

```
public static String fuzzyExiDecoder(String strinput, decodeMode
    dmode)
{
    String grammar = null;
    String result = null;

    grammar = GlobalValues.SCHEMA_PATH_MSG_BODY.toString();
    try {
        result = Exi2Xml(strinput, dmode, grammar);
    } catch (EXIException e1) {
        try {
            grammar = GlobalValues.SCHEMA_PATH_APP_PROTOCOL.toString
                ();
            result = Exi2Xml(strinput, dmode, grammar);
        } catch (EXIException e2) {
            grammar = GlobalValues.SCHEMA_PATH_XMLDSIG.toString();
            try {
                result = Exi2Xml(strinput, dmode, grammar);
            } catch (EXIException e3) {
                // do nothing
            } catch (Exception b3) {
                b3.printStackTrace();
            }
        } catch (Exception b2) {
            b2.printStackTrace();
        }
```

2. https://github.com/V2GClarity/RISE-V2G/tree/master/RISE-V2G-Shared/src/main/resources/schemas

```
    } catch (Exception b1) {
        b1.printStackTrace();
    }

    return result;
}
```

Listing 5. Fuzzy EXI data decoder.

This results in a tool we called *V2Gdecoder*, inherited by the shared library of *RISEV2G*, that exposes the following methods in a *dataprocess* class:

```
public class dataprocess {
    [...]
    public static String Xml2Exi(String xmlstr, decodeMode mode)
    [...]
    public static String Exi2Xml(String existr, decodeMode mode,
        String grammar)
    [...]
    public static String fuzzyExiDecoder(String strinput, decodeMode
         dmode)
    [...]
}
```

Listing 6. Exposed methods of V2Gdecoder.

This tool can be used in standalone, as well as a webservice to decode EXI or encode XML data:

```
$ java -jar V2Gdecoder.jar -h
usage: V2Gdecoder Helper
 -e,--exi             EXI format
 -f,--file <arg>      input file path
 -o,--output          output file path
 -s,--string <arg>    string to decode
 -w,--web             Webserver
 -x,--xml             XML format
```

Listing 7. V2Gdecoder helper.

So to perform data analysis while capturing packets, we need to extract V2GTP header from TCP packets, as shown in figure 17 and then use *V2Gdecoder* to be able to decode the V2GTP EXI payload as shown in listing 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns7:V2G_Message [...] xmlns:ns8="urn:iso:15118:2:2013:MsgHeader">
    <ns7:Header>
        <ns8:SessionID>41FE1835EEB99776</ns8:SessionID>
        <ns4:Signature>
            <ns4:SignedInfo>
```

```
                [...]
            </ns4:SignedInfo>
            <ns4:SignatureValue />
        </ns4:Signature>
    </ns7:Header>
    <ns7:Body>
        <ns5:ChargeParameterDiscoveryRes>
            <ns5:ResponseCode>OK</ns5:ResponseCode>
            <ns5:EVSEProcessing>Finished</ns5:EVSEProcessing>
            <ns6:SAScheduleList>
                <ns6:SAScheduleTuple>
                    <ns6:SAScheduleTupleID>1</ns6:SAScheduleTupleID>
                    <ns6:PMaxSchedule>
                        <ns6:PMaxScheduleEntry>
                            <ns6:RelativeTimeInterval>
                                <ns6:start>0</ns6:start>
                                <ns6:duration>7200</ns6:duration>
                            </ns6:RelativeTimeInterval>
                            <ns6:PMax>
                                <ns6:Multiplier>3</ns6:Multiplier>
                                <ns6:Unit>W</ns6:Unit>
                                <ns6:Value>11</ns6:Value>
                            </ns6:PMax>
                        </ns6:PMaxScheduleEntry>
                    </ns6:PMaxSchedule>
                [...]
```

Listing 8. Decoding V2GTP payload.

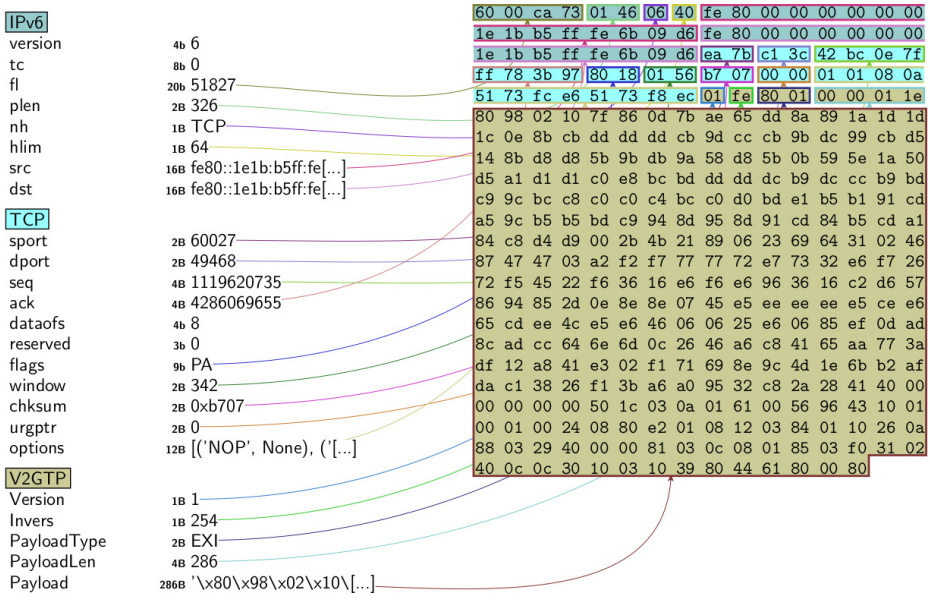

Fig. 17. V2GTP unkown data to decode.

The inverse operation can be performed by encoding a V2G XML file in EXI and encapsulating data into V2GTP → TCP → IPv6 → Ethernet before sending it to the target.

6 V2G Injector: Rise of the HPGPhoenix

All techniques, layers and tools have been assembled into one tool called *V2G Injector*. The resulting architecture of *V2G Injector* is shown in figure 18. Here are the available features:

— analyze V2GTP layer;
— extract EXI data;
— encode/decode data for V2G purpose;
— inject EXI data.

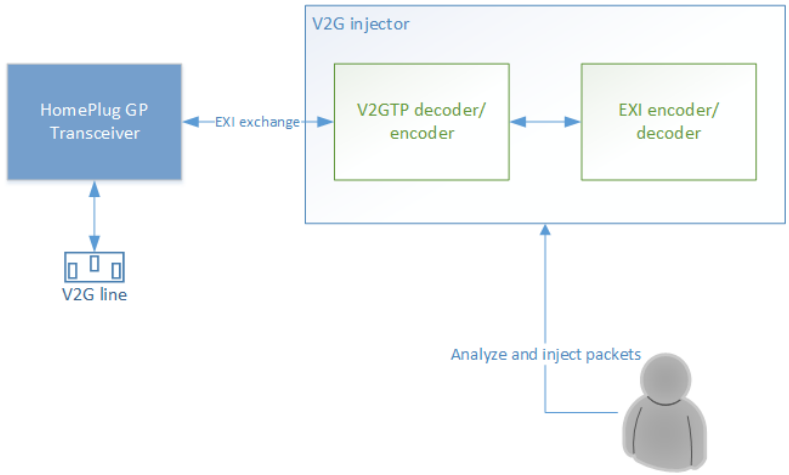

Fig. 18. Captured data exchange between a PEV and a EVSE.

6.1 Issues with missing grammar

During our tests, we also had issues when decoding V2G messages on few installations. Indeed, as mentioned earlier, it is important to have the right grammar, and it is common to see the use of old standards in the automotive industry such as “DIN 70121”, as shown in a *supportedAppProtocolReq* message:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns4:supportedAppProtocolReq xmlns:ns4="
    urn:iso:15118:2:2010:AppProtocol" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xmlns:ns3="http://www.w3.org/2001/
    XMLSchema">
    <AppProtocol>
        <ProtocolNamespace>urn:din:70121:2012:MsgDef</
            ProtocolNamespace>
        <VersionNumberMajor>2</VersionNumberMajor>
        <VersionNumberMinor>0</VersionNumberMinor>
        <SchemaID>0</SchemaID>
        <Priority>1</Priority>
    </AppProtocol>
    <AppProtocol>
        <ProtocolNamespace>urn:iso:15118:2:2013:MsgDef</
            ProtocolNamespace>
        <VersionNumberMajor>2</VersionNumberMajor><
        VersionNumberMinor>0</VersionNumberMinor>
        <SchemaID>1</SchemaID>
        <Priority>2</Priority>
    </AppProtocol>
</ns4:supportedAppProtocolReq>
```

Listing 9. Decoded EXI data.

As *RISE-V2G* does not provide an associated XSD for this namespace, we have to find them somewhere else. Thankfully, the *OpenV2G* project provides a C++ implementation that can be used to adapt XSD schemas of *RISE-V2G* to support DIN 70121, and we included it in *V2G Injector*.

Also, in case the namespace is not public and if the EVCC supports another protocol, it is possible to change protocol priorities during a Man-In-The-Middle attack to work with a grammar supported on our side. Otherwise, some work will be required to port the target grammar to *V2G Injector*.

7 Conclusion

This tool is the result of a long effort of understanding HomePlug GreenPHY and V2G communications as well as the behaviour of V2G modules. We hope this tool will help vendors and auditors to work in this specific area without spending weeks reading commercial specifications. The *V2G injector* provides a set of techniques and implements necessary features to be able to understand and inject data through the Control Pilot line, or through classic power plug if an installation shares the same network as the V2G units.

Intruding a network through the Power-Line has been shown as dangerous in domestic installation as an attacker could intrude the local

network of an individual or a company, but also extend the range of an attack. We have shown that, with the right tools, it is easy to intrude a V2G AVLN and interact with EVCC and SECC, analyse packets and craft them to attack different controllers and units, especially when data are exchanged in clear-text. But there is still an unexplored area with V2G units. Indeed, V2G units generally run complex systems, and can expose many interesting services like SSH/Telnet, FTP/SFTP, and/or (management) web services. Intruding such systems could potentially lead an attacker to use V2G units as a pivot to intrude other internal networks. So by opening this subject, we hope vendors and constructors will be more aware of potential risks and perform serious security assessments on SECC part as well as EVCC.

7.1 Future work

We are currently working on a domestic PLC, based on the QCA7k baseband, to interface to a V2C through a Power-Line but also with simple twisted pair lines to adapt it on IEC 62196 male/female connectors. This adaptation would reduce the cost from 150-200€ to approximately 30-50€.

Moreover, other features would be interesting to deploy:

— PEV + EVSE complete emulation and simulation;
— more EXI grammars;
— and automated fuzzing tests on EXI supported format.

References

1. Automobile propre. https://www.automobile-propre.com.
2. FAIFA: A first open source PLC tool. https://media.ccc.de/v/25c3-2901-en-faifa_a_first_open_source_plc_tool.
3. HomePlug AV Specification. https://www.homeplug.org/media/filer_public/61/c2/61c25a8b-0ef5-46ee-8fed-407dd6a650da/homeplug_av11_specification_final_public.pdf.
4. HomePlug GP Specification. https://www.homeplug.org/media/filer_public/74/40/7440ccd5-8c66-49ed-a2ce-5ef661932c27/homeplug_gp_specification_v111_final_public.pdf.
5. IEC 62196, Wikipedia. https://en.wikipedia.org/wiki/IEC_62196.
6. open-plc-utils. https://github.com/qca/open-plc-utils.
7. V2G Clarity, ISO 15118 manual. https://v2g-clarity.com/iso15118-manual/.
8. CHAdeMO Association. Technical Specifications of Quick Charger for the Electric Vehicle: CHAdeMO 1.0.1. *CHAdeMO Association: Tokyo*, 2013.
9. ISO technical committee. Road Vehicles–Vehicle to grid communication interface–Part 1: General information and use-case definition. *ISO Technical Committee: Geneva*, 2013.

10. German Institute for Standardization. Electomobility-Digital Communication Between a d.c. EV Charging Station and an Electric Vehicle for Control of d.c. Charging in the Combined Charging System. *German Institute for Standardization: Berlin*, 2014.
11. International Electrotechnical Commission. Communication networks and systems for power utility automationPart 90-8: IEC 61850 object models for electric mobility. *International Electrotechnical Commission: Geneva*, 2014.
12. International Electrotechnical Commission. Communication networks and systems for power utility automationPart 90-8: IEC 61850 object models for electric mobility. *International Electrotechnical Commission: Geneva*, 2014.
13. International Electrotechnical Commission. Electric vehicle conductive charging system-Part 23: DC electric vehicle charging station. *International Electrotechnical Commission: Geneva*, 2014.
14. International Electrotechnical Commission. Electric vehicle conductive charging system-Part 24: Digital communication between a d.c. EV charging station and an electric vehicle for control of d.c. charging. *International Electrotechnical Commission: Geneva*, 2014.
15. ISO technical committee. Road Vehicles–Vehicle to grid communication interface–Part 2: Network and application protocol requirements. *ISO Technical Committee: Geneva*, 2014.
16. ISO technical committees. Road Vehicles–Vehicle to grid communication interface–Part 3: Physical and data link layer requirements. *ISO Technical Committee: Geneva*, 2015.
17. Sébastien Dudek. HomePlugPWN. `https://github.com/FlUxIuS/HomePlugPWN`.
18. Sébastien Dudek. HomePlugAV PLC: practical attacks and backdooring. *NoSuchCon*, 2014.
19. Sébastien Dudek. PentHertz: The use of radio attacks during Red Team and pentests. *SecurityPWN*, 2018.
20. Michael Epping. Vehicle Charging Control Unit. *EMOB*, 2017.
21. Matthias Küdel. Design Guide for Combined Charging System. 2015.
22. Hyoseop Kim Minho Shin, Hwimin Kim and Hyuksoo Jang. Building an Interoperability Test System for Electric Vehicle Chargers Based on ISO/IEC 15118 and IEC 61850 Standards. `https://res.mdpi.com/applsci/applsci-06-00165/article\_deploy/applsci-06-00165.pdf`.
23. Sherman Gavette Ross Anderson Richard Newman, Larry Yonge. HomePlug AV Security Mechanisms. `https://www.cise.ufl.edu/~nemo/papers/ISPLC2007_AV_Security.pdf`.
24. Siemens. OpenV2G. `https://github.com/Martin-P/OpenV2G`.
25. V2GClarity. RISE-V2G. `https://github.com/V2GClarity/RISE-V2G`.
26. Peng Wang Zhigang Ji Wenpeng Luan, Gen Li. Security of V2G Networks: A Review. *Boletín Técnico, Vol.55, Issue 17*, 2017.
27. Yan Zhang and Stein Gjessing. Securing Vehicle-to-Grid Communications in the Smart Grid. *IEEE Wireless Communications*, 2013.