Key Management:

How do we get trust and security into a puzzle game?

Bennet Blischke, April 2022

Prerequisite:

We are building an IoT, wireless connected, puzzle game. We have in mind that the puzzle game can be a product which might be sold to non technical personal and organisations.

In the following these words are defined as:
- Node: An 802.15.04 connected embedded device running RIOT-OS + Application
  - -> Devices that belong to the game
- Communication participant: A devices connected to the 802.15.04 network
  - -> Including devices that might not belonging to the game
- Operator: An administrator with access to the management web application of the game

---

Requirements[R] for the key management, security and authenticity:

1. Works on top of 802.15.04
2. An easy solution is preferable over a complicated one - it's a puzzle game not a banking app
3. New IoT nodes can be added to the puzzle game via Plug & Play
4. Plug & Play can be done at runtime of the game
5. Plug & Play is easy enough that non-developers can do it successfully
6. Security is provided as in:
   1. Protection against eavesdropping of potentially sensitive data
   2. Protection against replay attacks
   3. Protection agains injection of foreign data
7. Authentication is provided as in:
   1. Nodes are known and trusted
   2. Only the game operators can add new nodes to the game
   3. Unauthenticated nodes can not impose trusted nodes
   4. Other communication participants can't operate or use the games infrastructure

---

Solution proposal

- Using DTLS the requirements [R 6.1, 6.2, 6.3] can be solved
- Asymmetric cryptography can solve [R 7.1, 7.2, 7.3, 7.4]
- For solving the Plug & Play requirements [R 3, 4, 5], an additional onboarding process is required
- The onboarding process, of a new node into the game, must be simple and highly automated

These technologies and processes create further challenges:
- In order to speak DTLS, two nodes must already have shared some kind of trust e.g. PSK
- Asymmetric cryptography requires management of public and private keys
- During an onboarding, the new node must be authenticated by exchanging and establishing trust with the game

Attack vectors[AV] present in the proposed solution

1. The onboarding could be done by an attacker, gaining trust & authentication
2. If the onboarding process is too accessible or done via insecure means, an attacker could eavesdrop on it and potentially replay it later
3. When a PSK is used for DTLS, an attacker could steal it and use it to establish a DTLS communication within the game. Depending on the DTLS implementation, even decrypt traffic of other nodes
4. If public & private keys are used for DTLS, a stolen private key could be used to impose the original owner as well as all effects of 2.
5. If authentication is done via public & private keys, a stolen private key could be used to authenticate an attacker as an original node
6. If keys or other forms of trust are stored within a node, an attacker could steal a node and extract it
7. If the trust or keys are the same for all puzzle games, one compromised game compromises all

Final solution

- Every node has its own public and private key which are generated at build time or time of flashing. This ensures that one stolen key pair does not compromise all games [AV 7]

1. When a node is powered on, it joins the games 802.15.4 network as a communication participant (without using DTLS, unauthenticated)
2. The operator is informed via the web application that a new node has joined the network and must be authenticated before it can join the game [R 5]
3. The operator plugs the new node into the games gateway via USB [AV 2]
4. The gateway asks the operator via the webapp whether the newly connected node is known and whether it should be trusted [R 6][AV 1]
5. After the operators acknowledgement, the gateway and the new node exchange their public keys and the node is now trusted and authenticated
6. The new node can now speak DTLS with the gateway using the exchanged keys [AV 3]
7. The gateway regularly e.g. 15 min interval confirms that all trusted nodes are still connected to the network and reachable
    1. If a node is not reachable, it is considered stolen and ejected from the game. The public key of this node can no longer be trusted. [AV 6, transitive 4, 5]
    2. The node needs to be restarted and re-authenticated, see step 1

Remaining security issues

These remaining issues are considered out-of-scope for this project

- If an attacker steals a node, extracts the keys and then reinserts the node into the network before the gateway's verification interval expires (=> the node is not detected as stolen), the attacker has undermined the security and authenticity of the game
    - This can be made significantly more unlikely by:
        - shortening the verification interval
        - Enable flash read-out-protection in the SoC
        - Storing the gateways public key in volatile memory which will be lost in case of a power failure
- Implementation bugs and security holes
    - It is helpful to follow the common best practices in the development and enabling all possible security features e.g. stack smashing protection
- A stolen node could be tempered and sold as secondhand to undermine the security of a specific operators game
    - Additional hardware trust, secure boot and other options could be explored

What's next?

- The specific type of asymmetric cryptography must be selected and agreed on
- The gateway needs its own, static pair of keys, either at build time or during boot of the device
- The generation of asymmetric keys at build or flash time needs to be specified and implemented
- The onboarding process is vague:
    - Software must be written which hooks on USB events on the gateway, communicates with the web app and the node, exchanges the keys
    - The interaction with the web app must be specified
- The keys stored in the gateway must be run-time accessible by the gateway software