draft-dvir-roll-security-authentication-01
and
draft-dvir-roll-security-key-agreement

Amit Dvir
Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics

this is joint work with

Levente Buttyán, Tamás Holczer and Dóra László

RPL – WG Vs CrySyS Security View

Both are important and should be implemented.

- A security framework.
- Link-by-link protection.
- External attacker, the basic assumption is trustful nodes.

WG

- After a while, a node in the field can be the problem
- Nodes may be accessed physically/logically and get compromised.
- Internal attacks by compromised nodes.

CrySys

Example

Reliability

- Availability
- Connection oriented services
- Quality of service

Industrial Automation

Homeland Security

Safety

- Verified system properties
- Quality of service

Telemedicine

Security

- Data integrity
- Secrecy
- Node integrity
- Privacy

Duration: 1/1/09 – 31/12/12,

Total effort: 370 PM, Total costs/funding: 4.0 / 2.8 Mio EUR

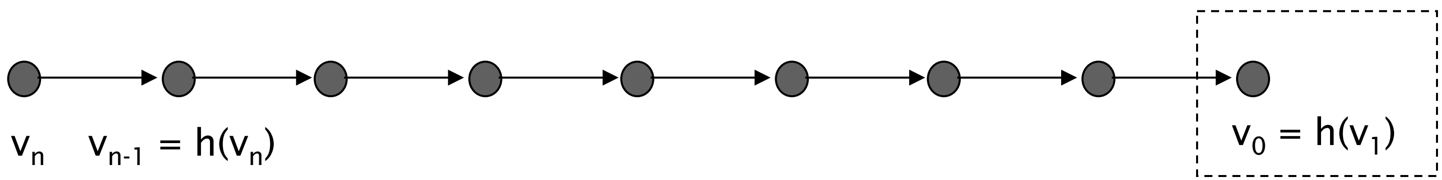
The review went well

Security/Efficiency Trade off (or why protect some and not all)

- An internal adversary can attack in many different ways.
- Not all attacks may have a major influence.
- Reconstructing the entire DAG, exhaust the nodes' batteries, or eavesdropping a large part of the traffic can have major or even critical influence.
- One way to achieve attacker' goals is to change/modify the Version Number or to change/modify the DODAG node Rank.
  - Modifying these may have a global effect.
- Therefore, in this draft we present a security scheme to prevent those attacks.

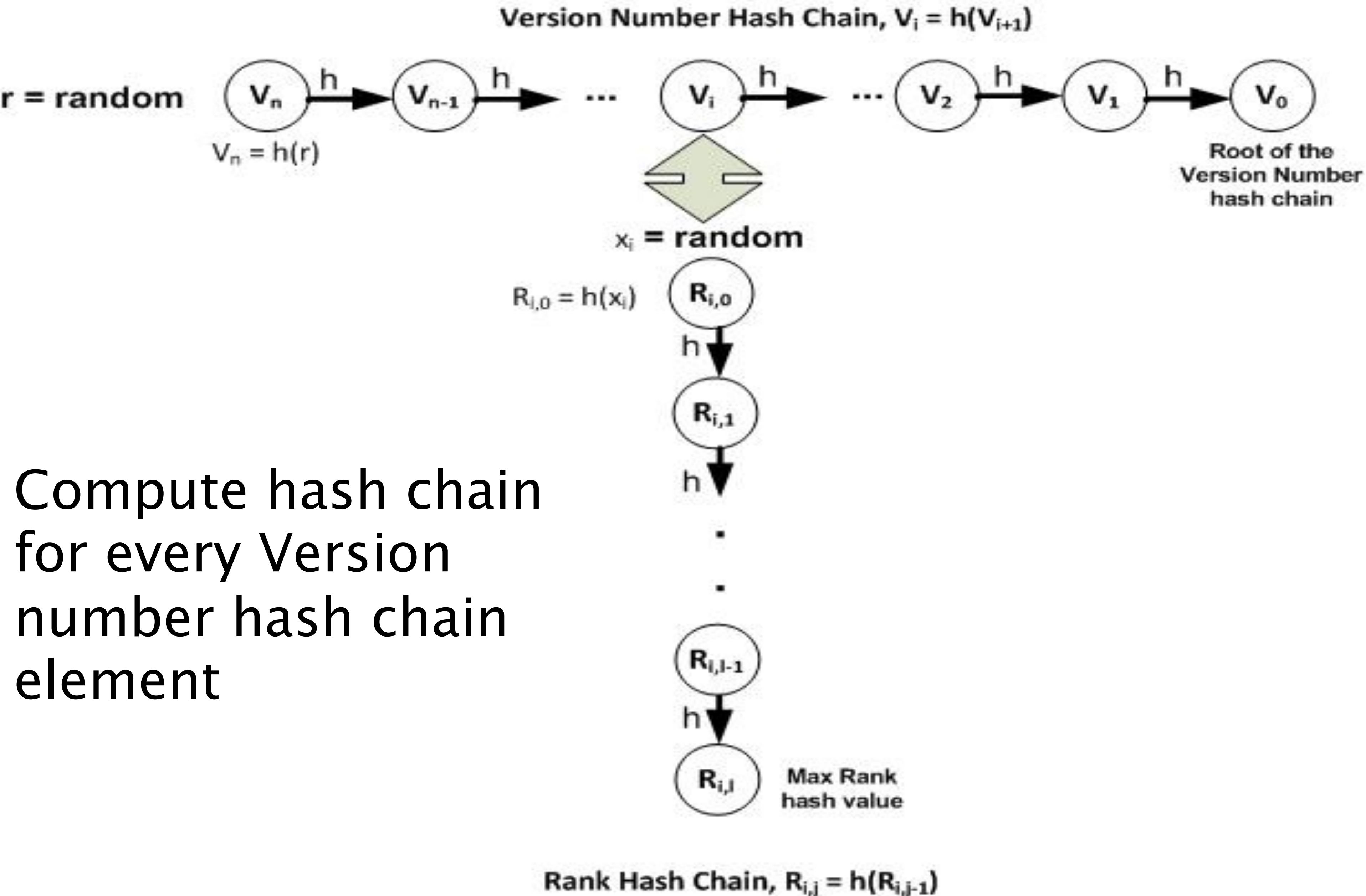
Version Number authentication

- The DODAG root generates a random number v_n, and **computes a hash chain** $v_{n-1} = h(v_n)$, $v_{n-2} = h(v_{n-1})$, …, $v_0 = h(v_1)$

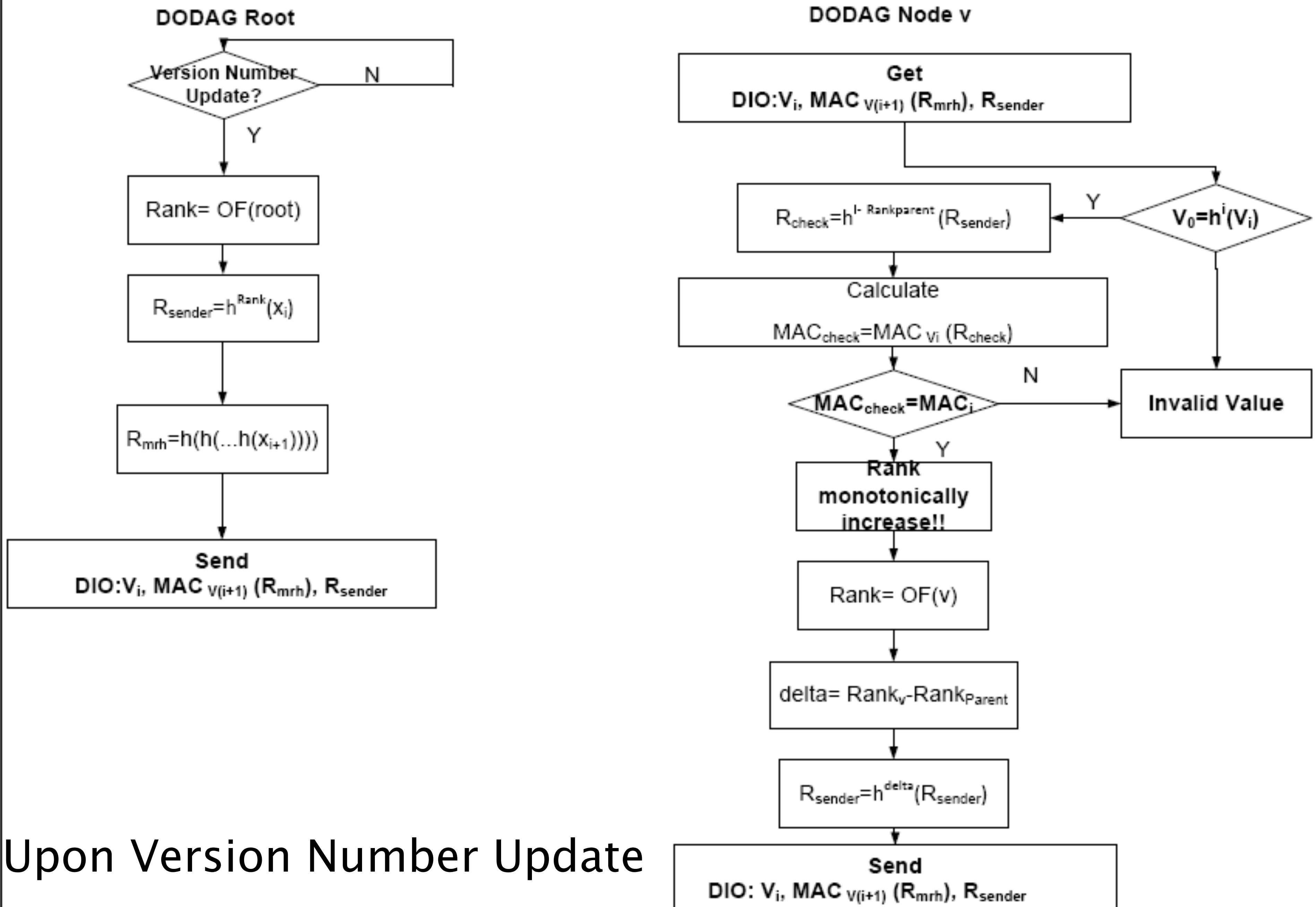


- The DODAG root **distributes the root v_0** of the version hash chain to all nodes in the network by
  - Including v_0 in a DIO message
  - Authenticating v_0 and the static fields of the DIO message, such that all other nodes can be sure that v_0 originates from the DODAG root
    - **Digital signature** verifiable with the public key of the DODAG root
  - When the DODAG root sends out a DIO message with a new Version Number, it also releases the next version hash from the chain
    - Note that the next version hash v_i cannot be computed from the last released value $v_{i-1} = h(v_i)$ due to the one-way property of the hash function h

Rank Authentication



Compute hash chain for every Version number hash chain element



Upon Version Number Update

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=10     | Option Length |Code |  Flags  |   Algorithm   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
:                      Authentication Data                      :
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
+-----+--------------------------------+
|Value|        Code Value Type         |
+-----+--------------------------------+
|  0  |Version Number Hash Chain Value|
|     |                                |
|  1  |Version Number Hash Chain Root |
|     |                                |
|  2  |Rank Hash Chain Value           |
|     |                                |
|  3  |MAC of Max Rank Hash Value      |
|     |                                |
|  4  |Integrity Protection Value      |
|     |                                |
| else|Unassigned                      |
|     |                                |
+-----+--------------------------------+
```

Figure 6: Code Type

Pairwise key establishment – LEAP++

- **Different draft**
- main assumptions:
  - Any node will not be compromised within T time after its deployment
  - Any node can discover its neighbors and set up keys with them within $T_{kex} < T$ time
    - typically, T_{kex} is a few seconds, so these assumptions make sense in practice
  - Each node has a preshared secret key K at boot/restart
- protocol phases:
  - key pre-distribution phase
  - neighbor discovery phase
  - link key establishment phase
  - key erasure phase

Pairwise key	**Hello Message - Any RPL message.** **Response Message - Unicast DIS.** **Ack Message (OPTIONAL) - Unicast DIO**

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Hello Message, u -> *:  (u)
Response Message, v -> u: (MAC(Kv, v|u))
Ack Message, u -> v: (MAC(Kuv, u|v))
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Cluster Key	**Hello Message - Any Unicast RPL message.** **Ack Message (OPTIONAL) - Any Unicast RPL message**

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ENC Message, u -> *:  (ENC_Kuv(r))
Ack Message, v -> u:  (MAC(Kuv, r))
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Conclusion

- We identified illegitimate Version Number increases and Rank value decreases as two powerful attacks against RPL.
- We proposed solutions that prevents both of the attacks based on stable mechanisms.
- We use public/private operations once in a long while; we use nearly only symmetric operations.
- For the key exchange, we are using preinstalled keys for a short time.

Answers:

- In theory, *l* should be 65535 to have a different value for each possible Rank value (Rank is 16 bits long [I-D.ietf-roll-rpl]). In practice, 256 is large enough as for most of the operations DAGRank [I-D.ietf-roll-rpl] is used (the DAG Rank of a node is the upper 8 bits of the Rank), which is 8 bits long.
  - If DAGRank is used when defining the hash chain, then all occurrences of Rank must be substituted by DAGRank in the sequel

- **Elliptic Curve Cryptography**
  - ECC-SECP256K1 with SHA-256