Node Security

Ganeticon, Sep 2014

Helga Velroyen, helgav@google.com

Agenda

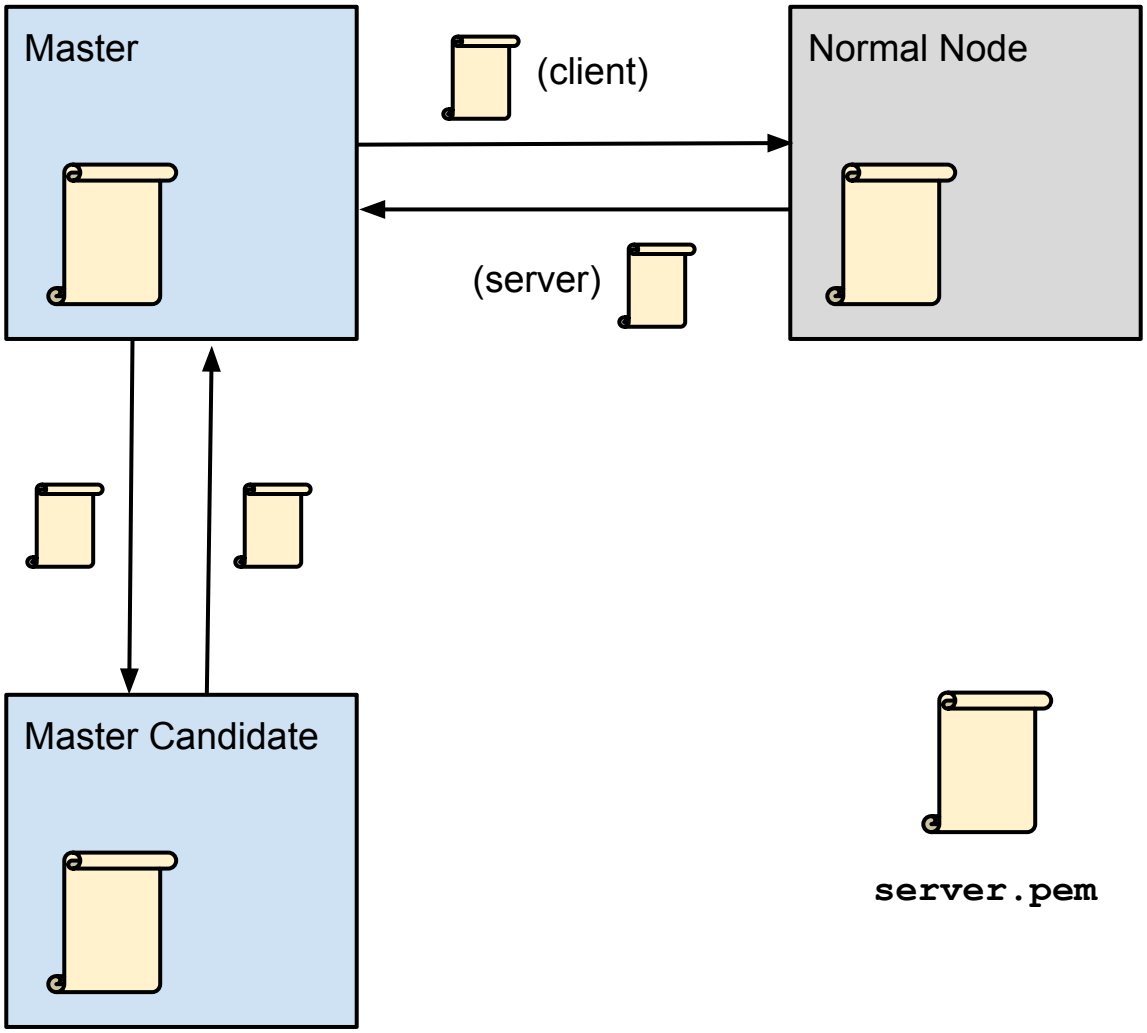
- SSL node security
- SSH node security

Google Confidential and Proprietary

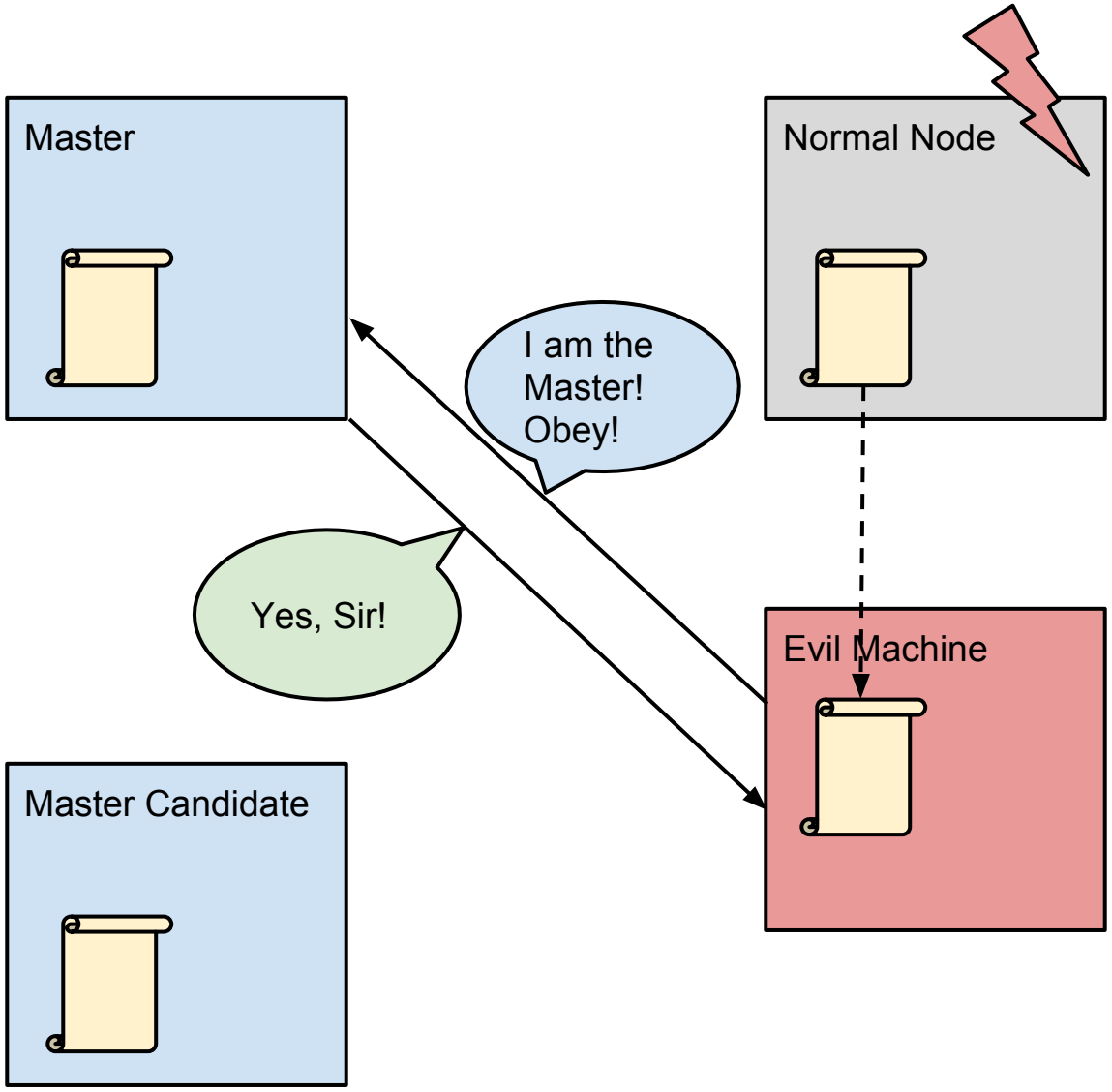
SSL

Node Security

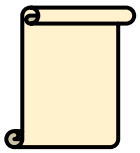
RPC via SSL (pre 2.11)



Google Confidential and Proprietary



Compromising a Node can lead to random Nodes impersonating the Master



`server.pem`

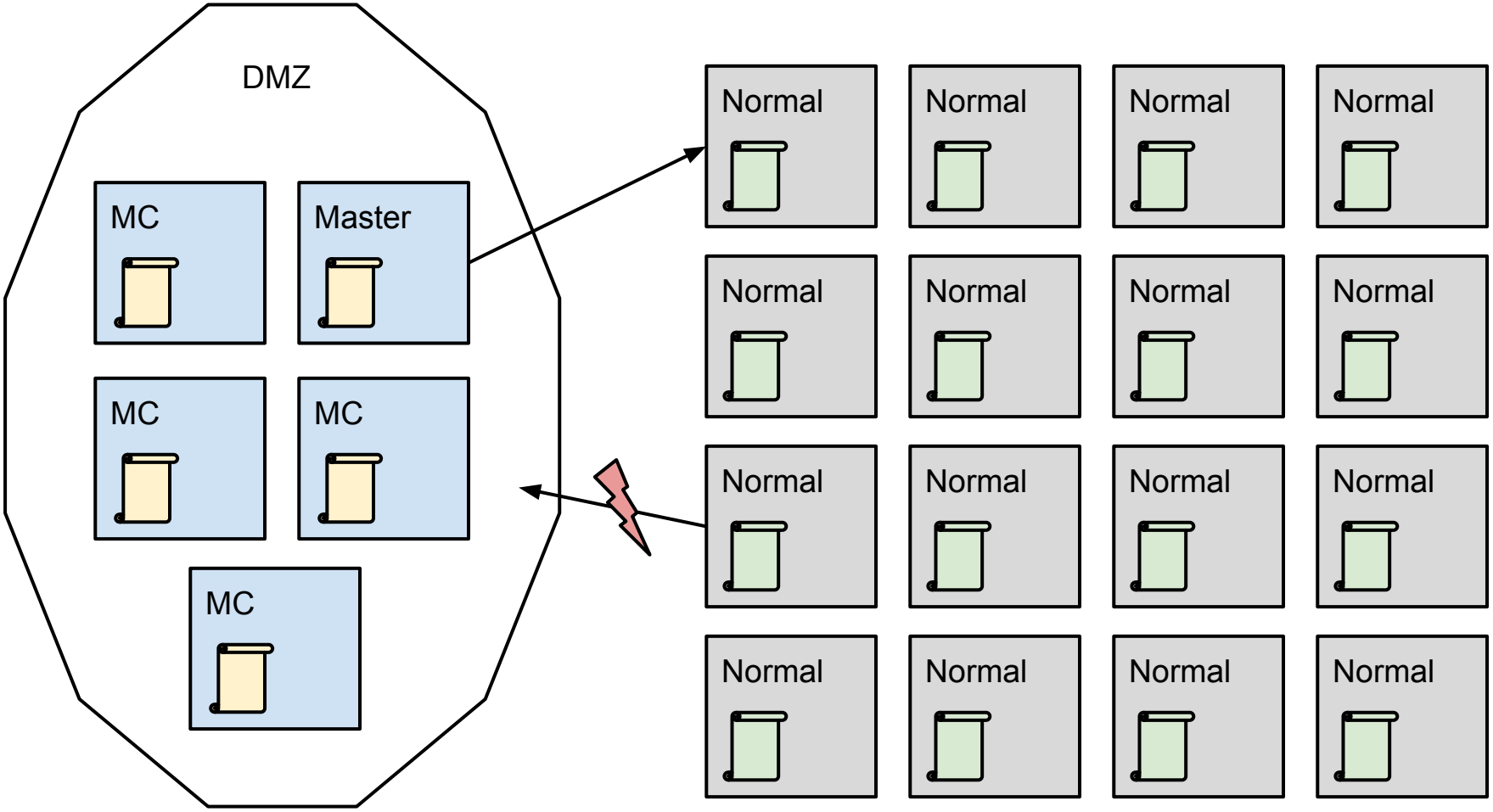
Google Confidential and Proprietary

Design Goals

- A compromised normal node should not harm other nodes
- Nice to have: it should be possible to identify and remove a compromised master or master candidate
- Details: doc/design-node-security.rst

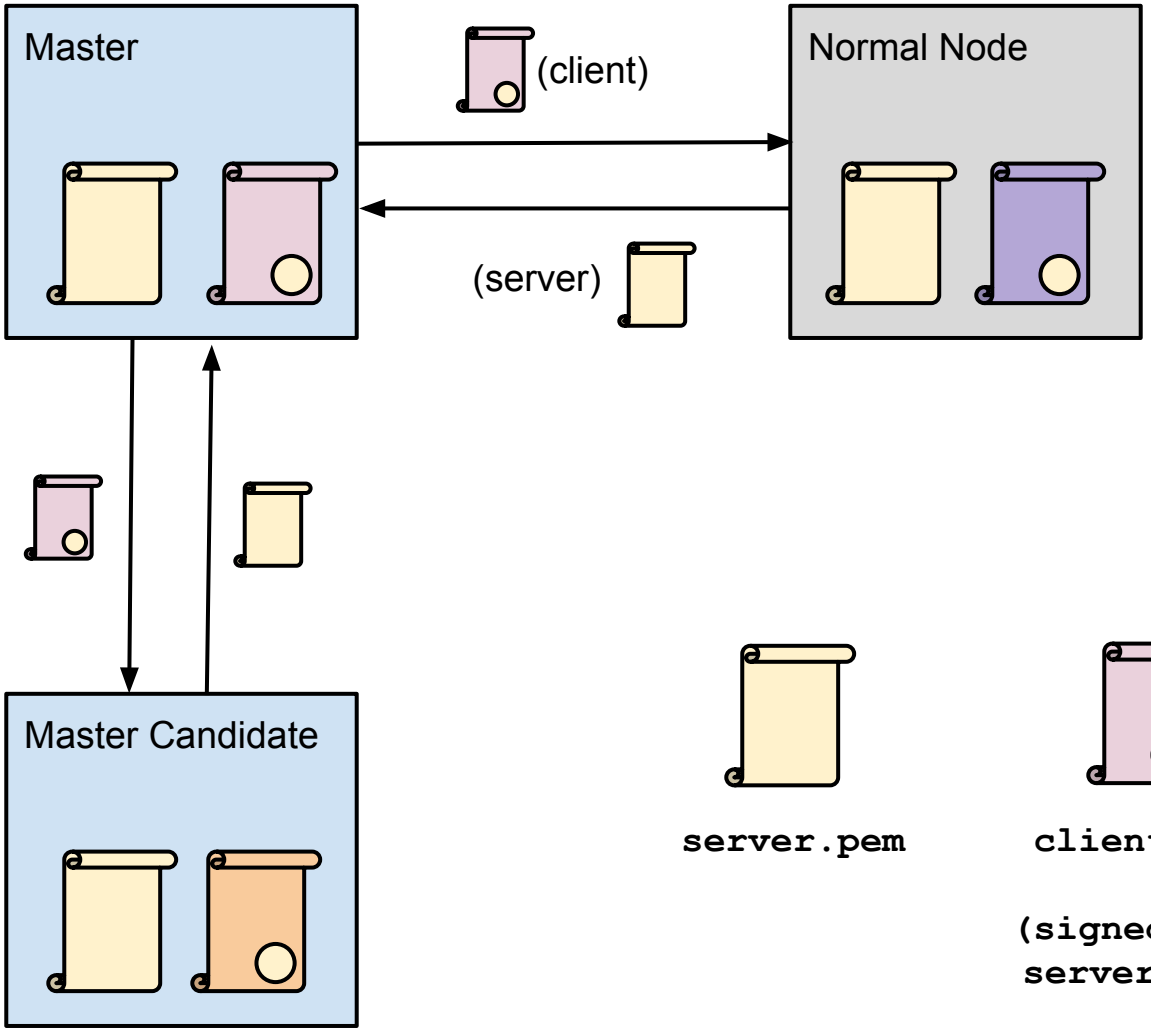
Google Confidential and Proprietary

Possible Setup: Shield your Master Candidates

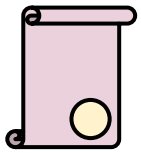


Google Confidential and Proprietary

RPC via SSL Individual Client Certificates



`server.pem`

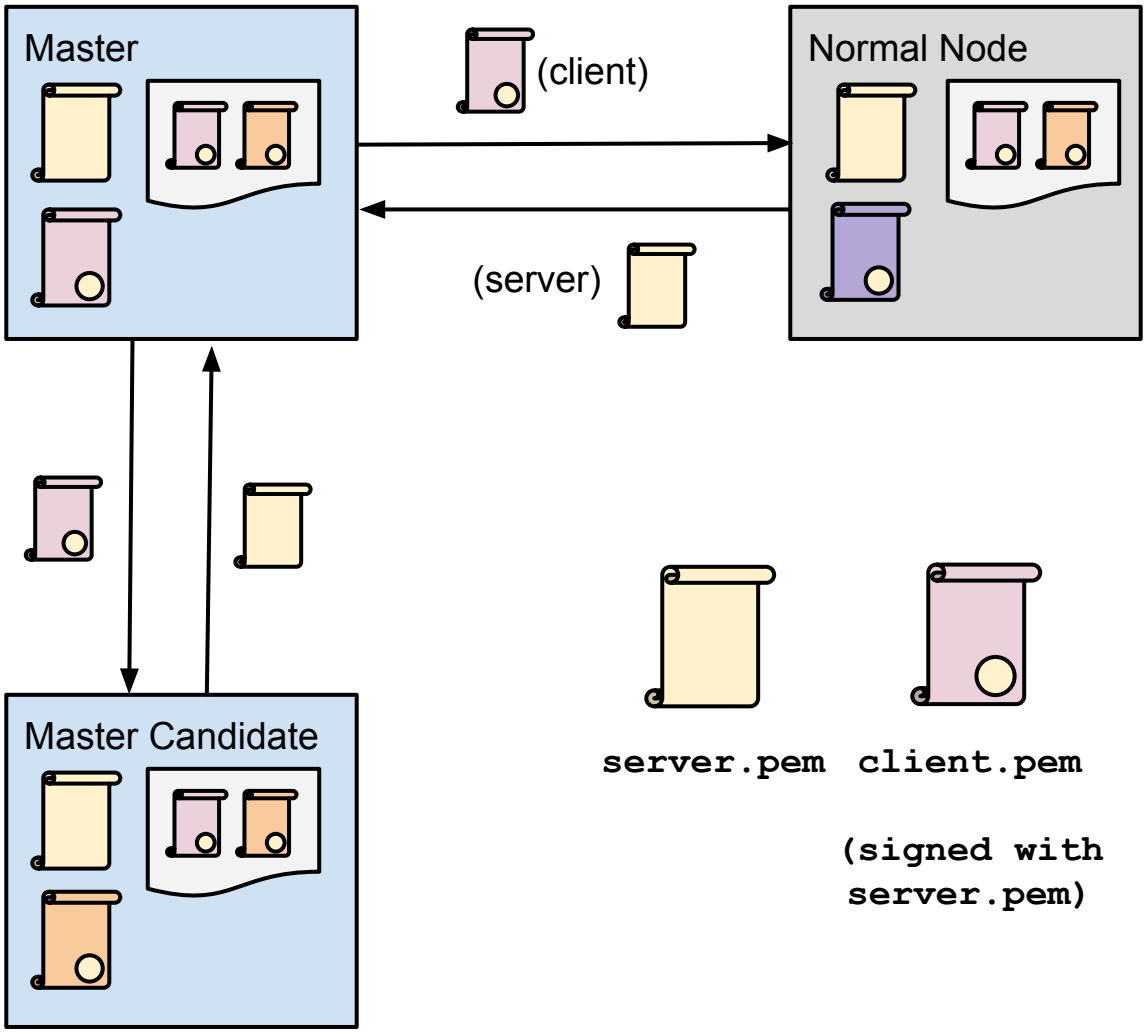


`client.pem`

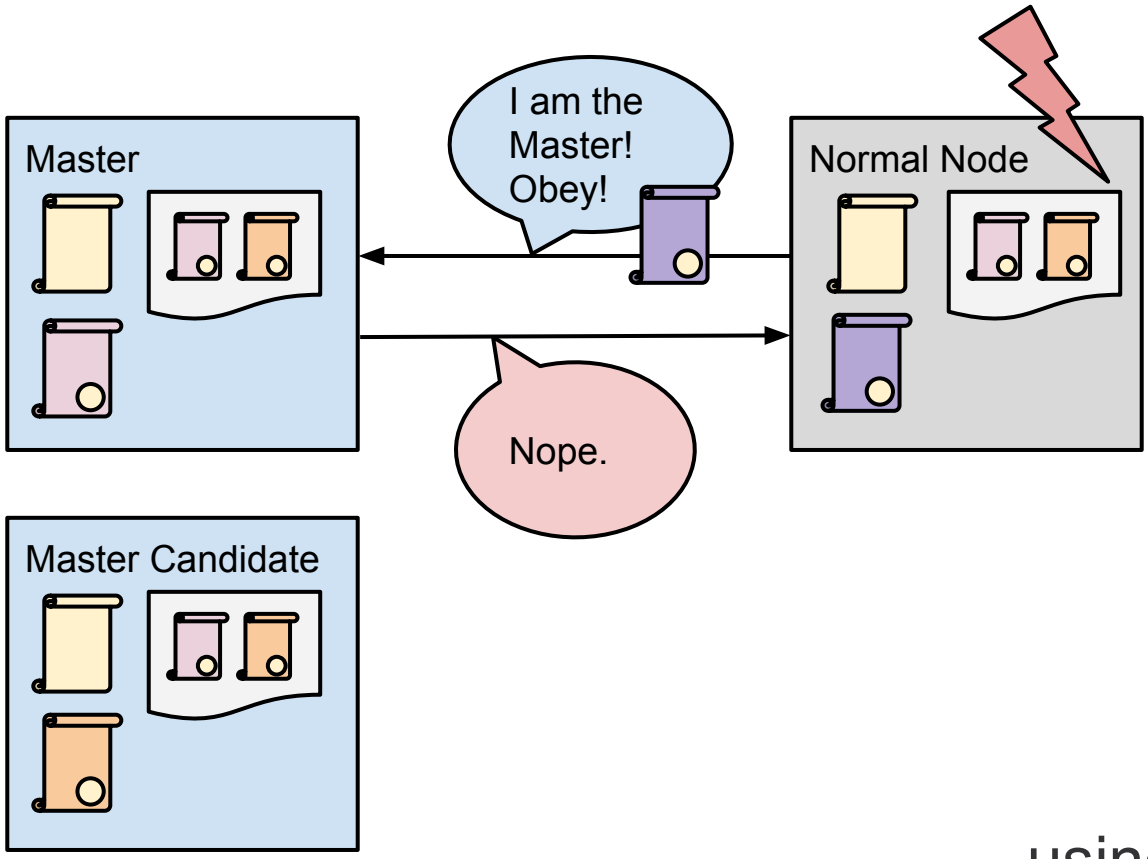
`(signed with server.pem)`

Google Confidential and Proprietary

RPC via SSL Master Candidate Certificate Lists

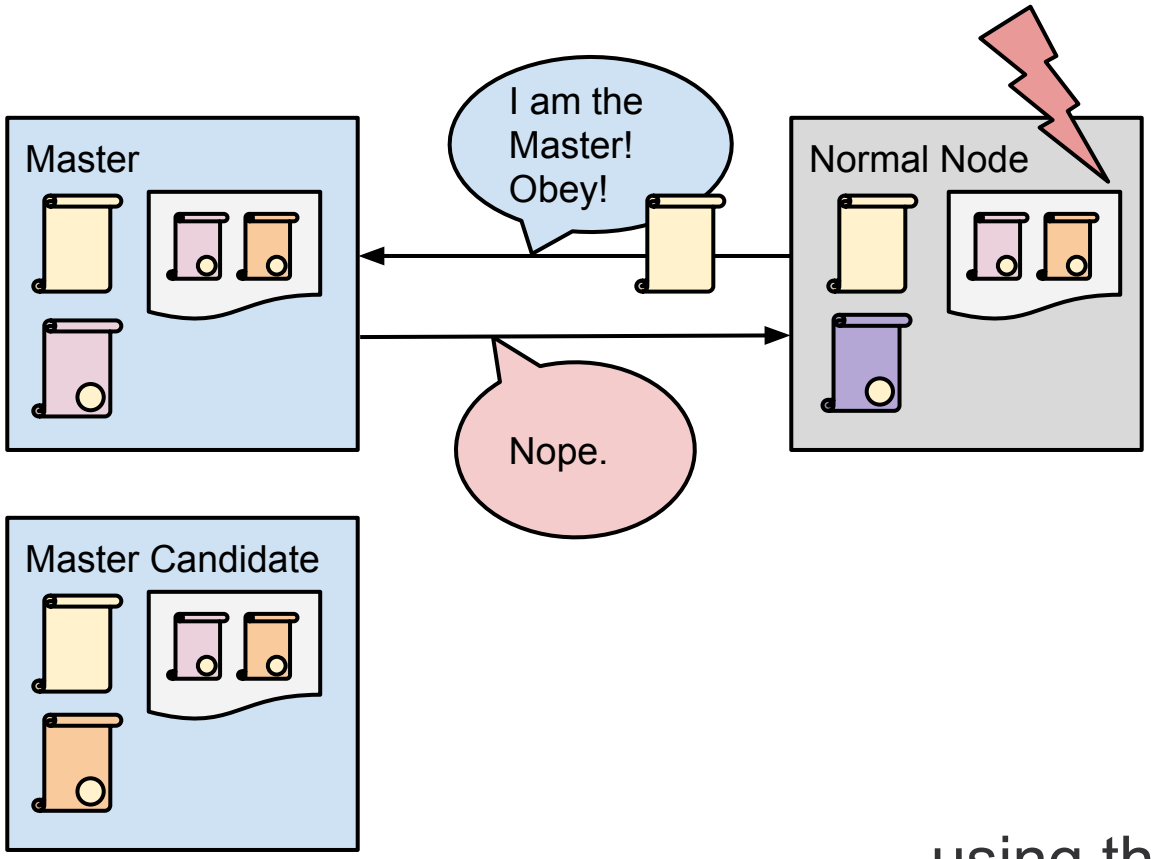


Google Confidential and Proprietary



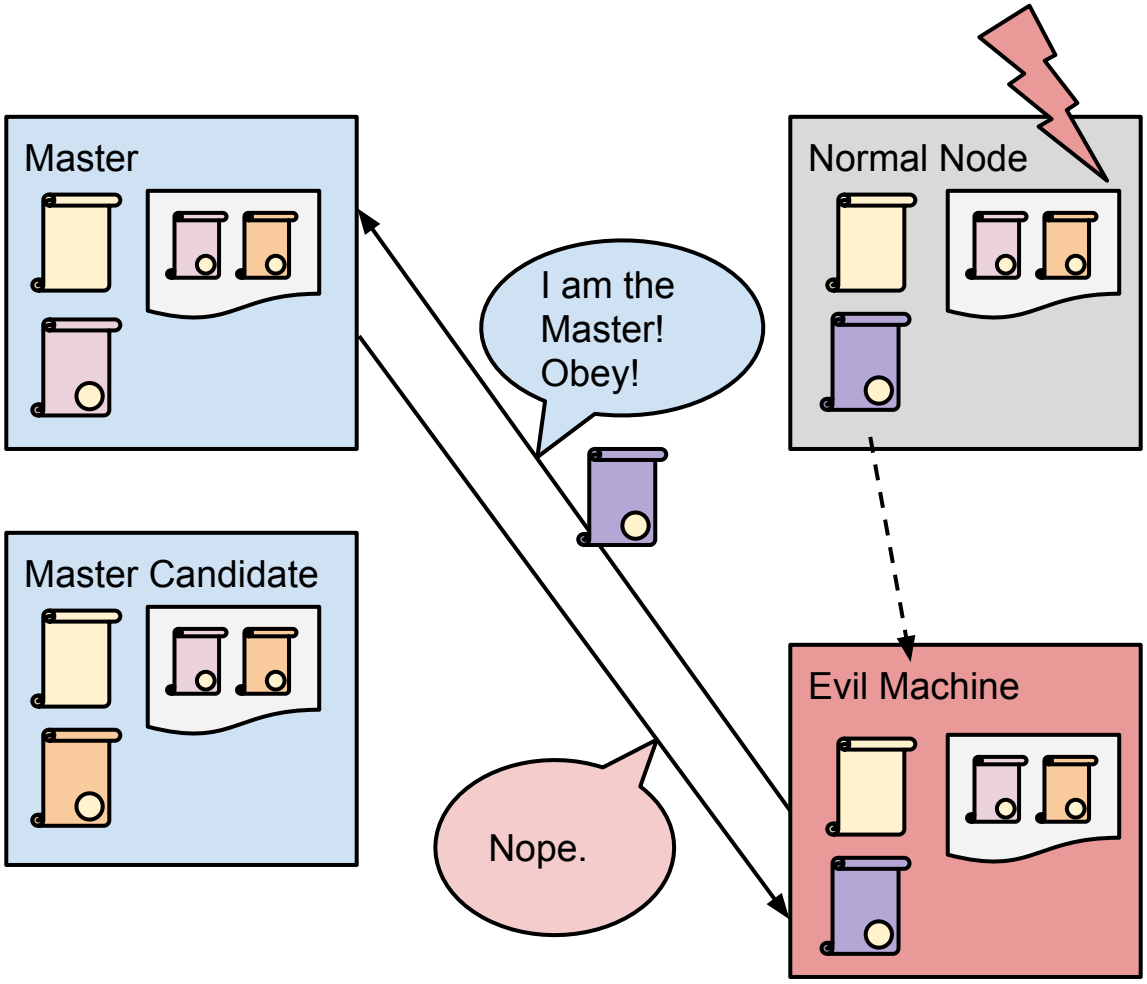
Attack Scenario:
using a client certificate
of a normal node is futile.

Google Confidential and Proprietary



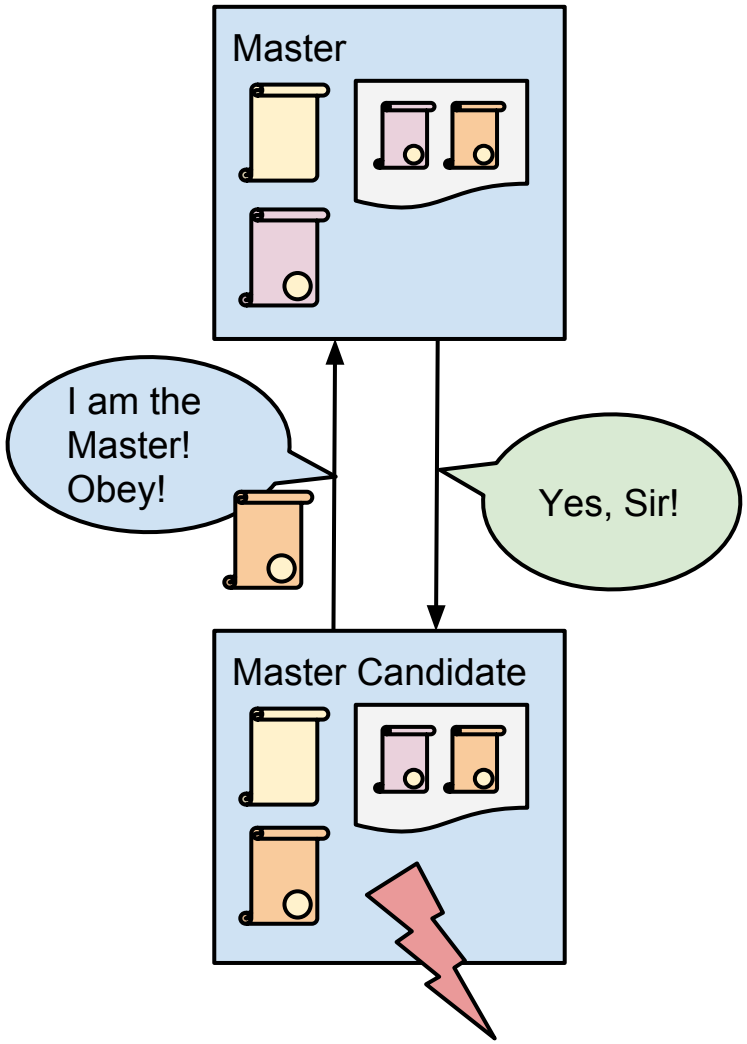
Attack Scenario:
using the **server** certificate is futile.

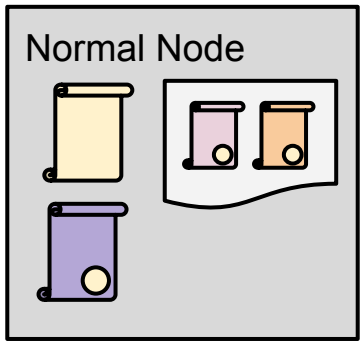
Google Confidential and Proprietary



Attack Scenario: Same applies to external machines using copied certificates

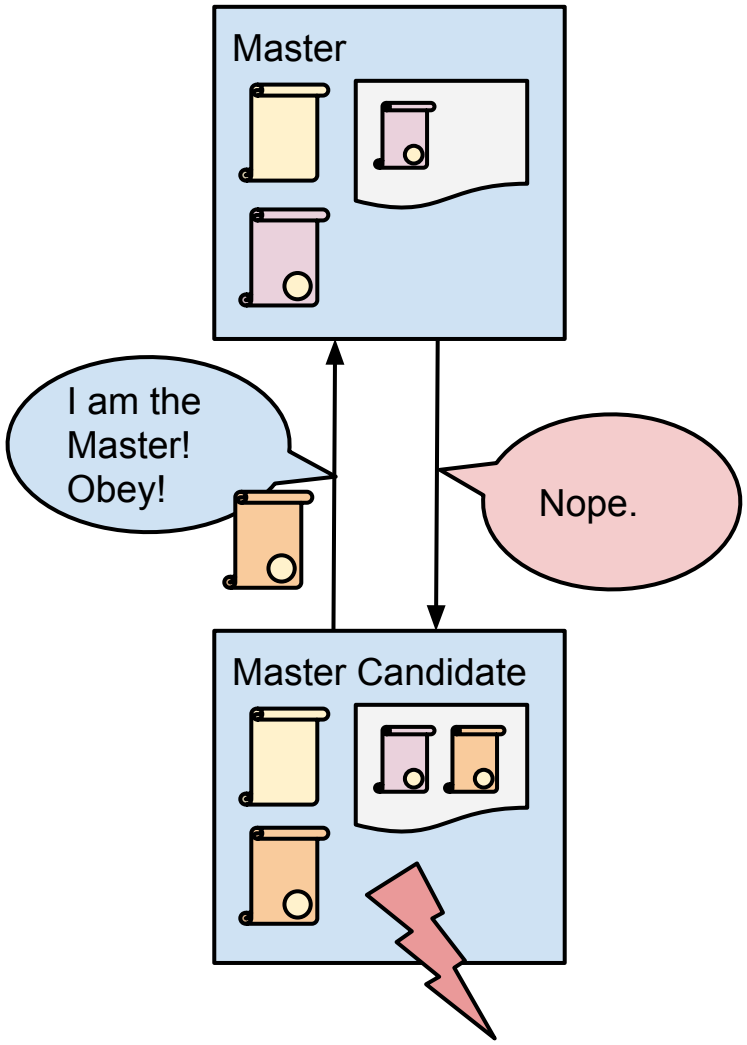
Google Confidential and Proprietary

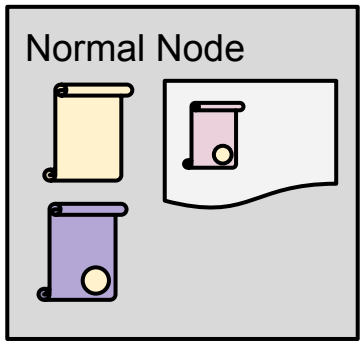




Attack Scenario:
Compromised
Master Candidates
can still own the cluster :(

Google Confidential and Proprietary





But at least a certificate of an identified compromised master candidate can be removed

Google Confidential and Proprietary

Implementation Challenges

- Bootstrapping: how to change from “Before” to “After” on upgrading Clusters
- Cluster renew-crypto
- Writing Configuration + Timing Issues
- Fun with undocumented differences between SSL libraries

Google Confidential and Proprietary

Comparison: Before and After

- Poor man's usage of SSL
- One certificate for client and server role
- The same certificate for all nodes
- Certificate self-signed
- One compromised node can compromise the cluster

- Slightly richer man's usage of SSL
- One shared certificate used for server role
- Individual certificates used for client role
- Only master candidate certificates allowed as client
- A compromised normal node cannot harm other nodes
- A compromised master candidate can still harm, but certificate can be removed individually

Google Confidential and Proprietary

Can we do better?

Yes! No? Maybe?

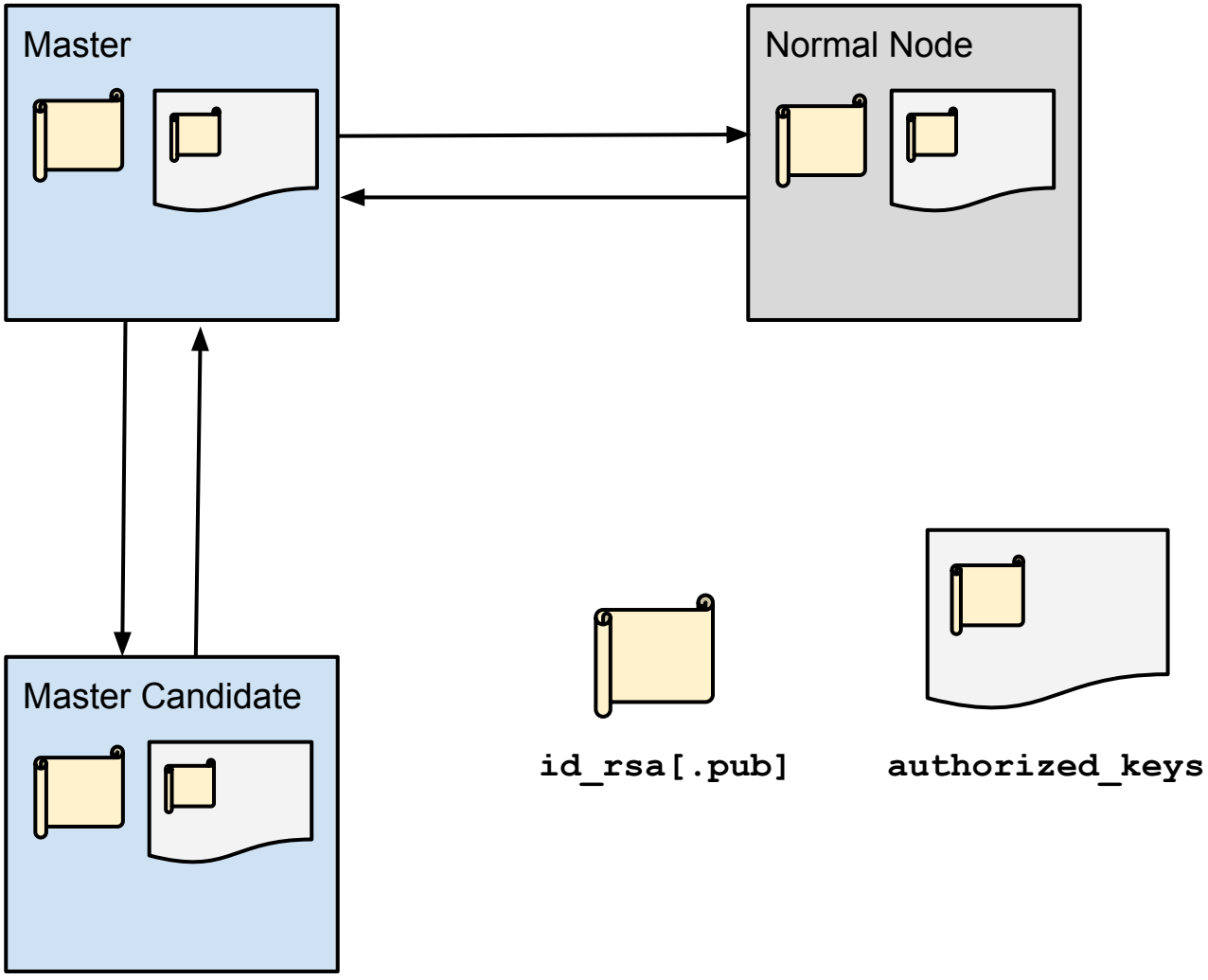
Discuss this with us on Thursday in the Security Design session!

Google Confidential and Proprietary

SSH

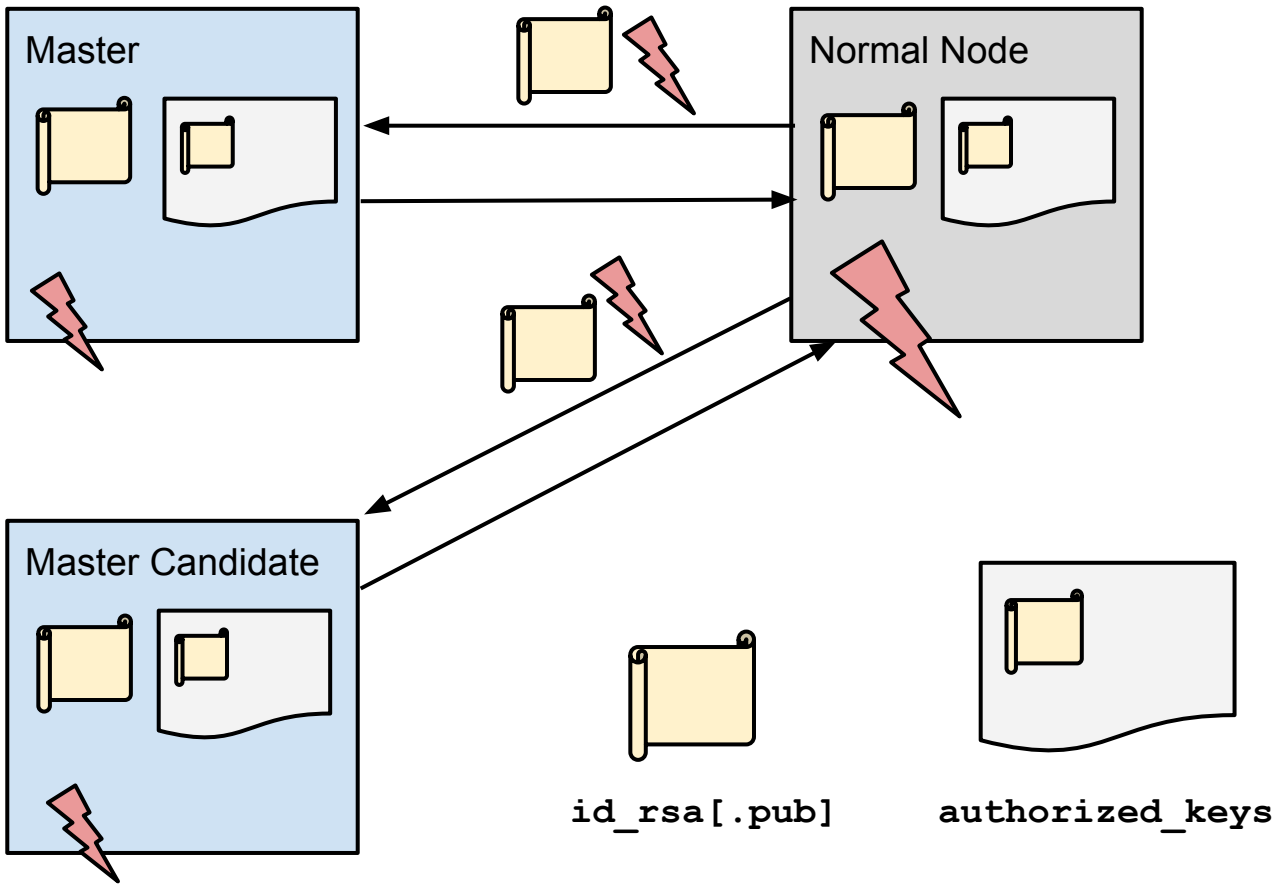
Node Security

SSH key setup (pre 2.13)



Google Confidential and Proprietary

Attack Scenario: One Node owns it all



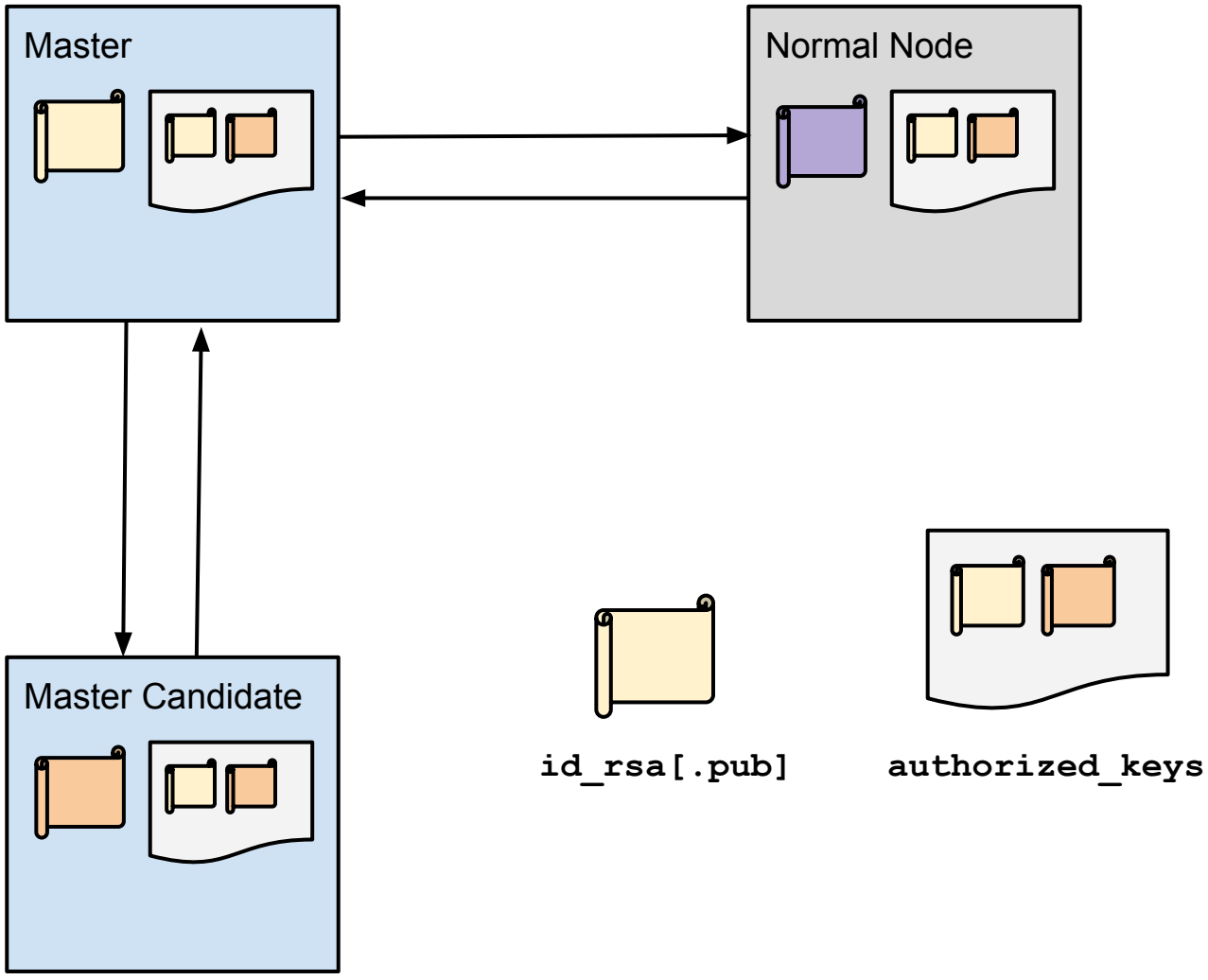
Google Confidential and Proprietary

Design Goals

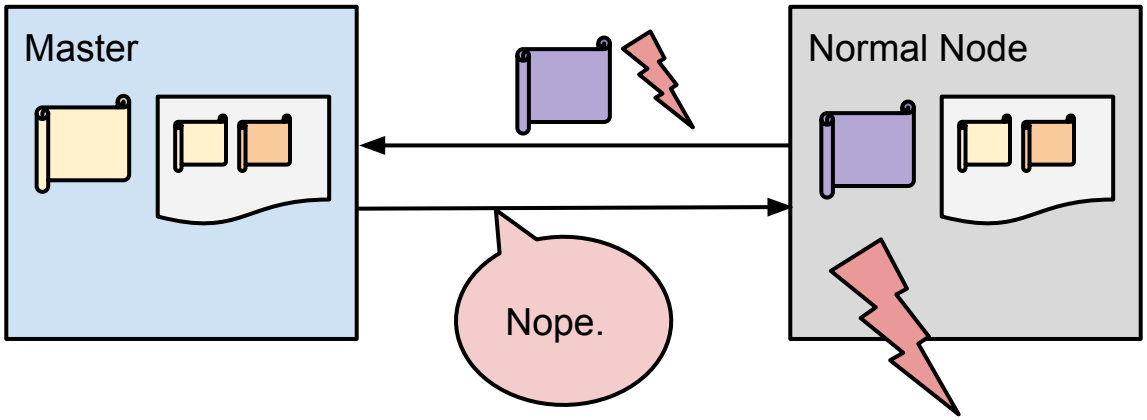
- Same as for SSL actually
- Compromised normal nodes should do no harm to others
- Nice to have: Compromised master candidates can be removed separately
- Details: doc/design-node-security.rst

Google Confidential and Proprietary

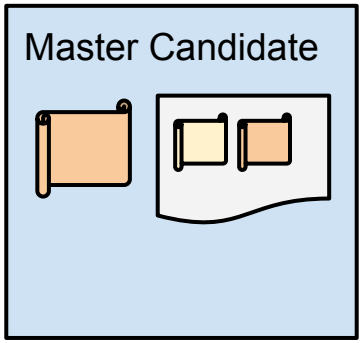
SSH key setup in 2.13 (hopefully)

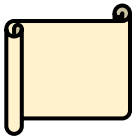


Google Confidential and Proprietary

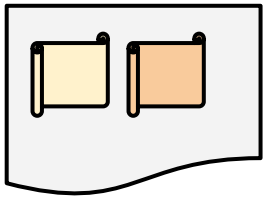


Attack Scenario: Compromised normal node does no harm



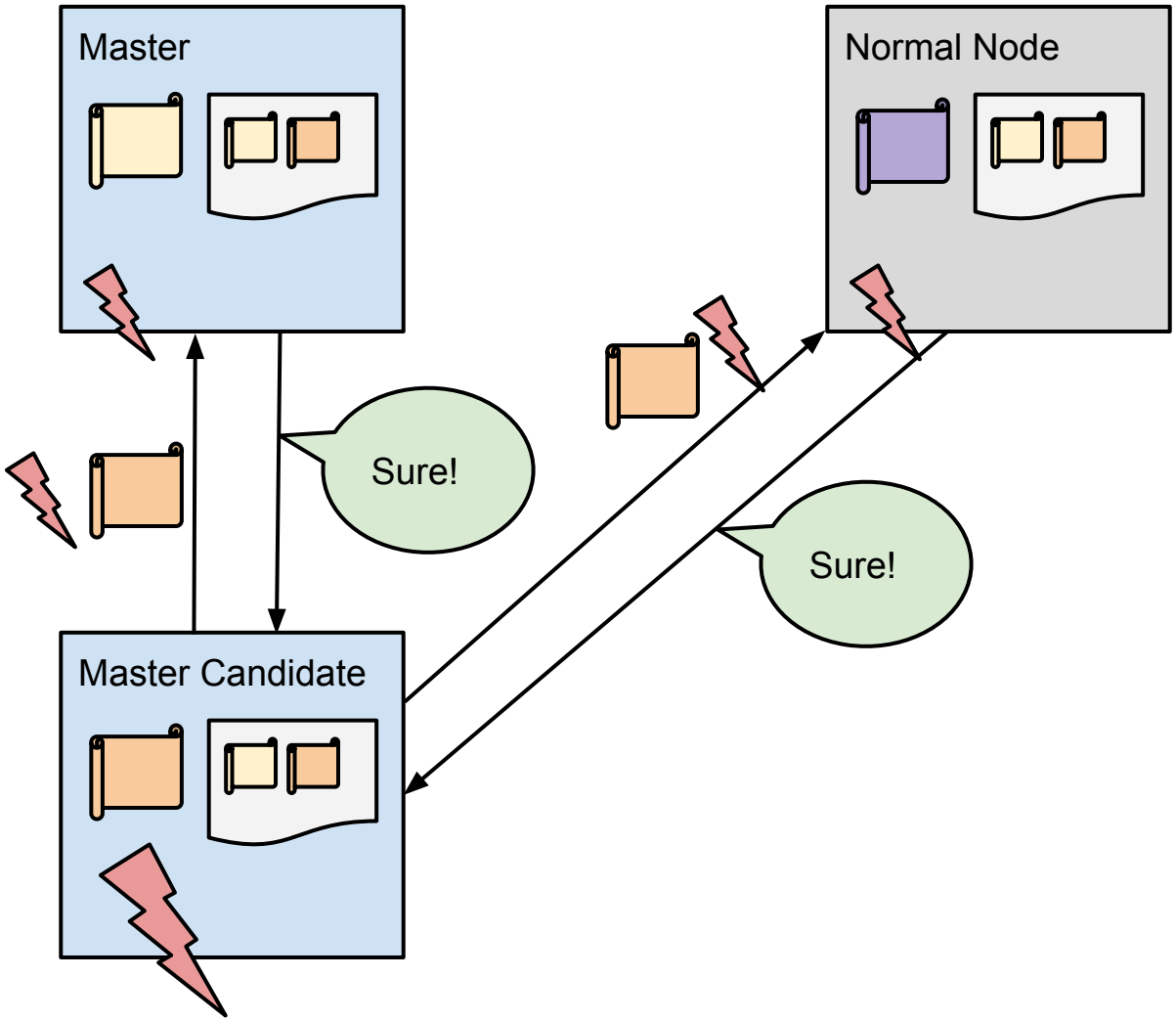


`id_rsa[.pub]`



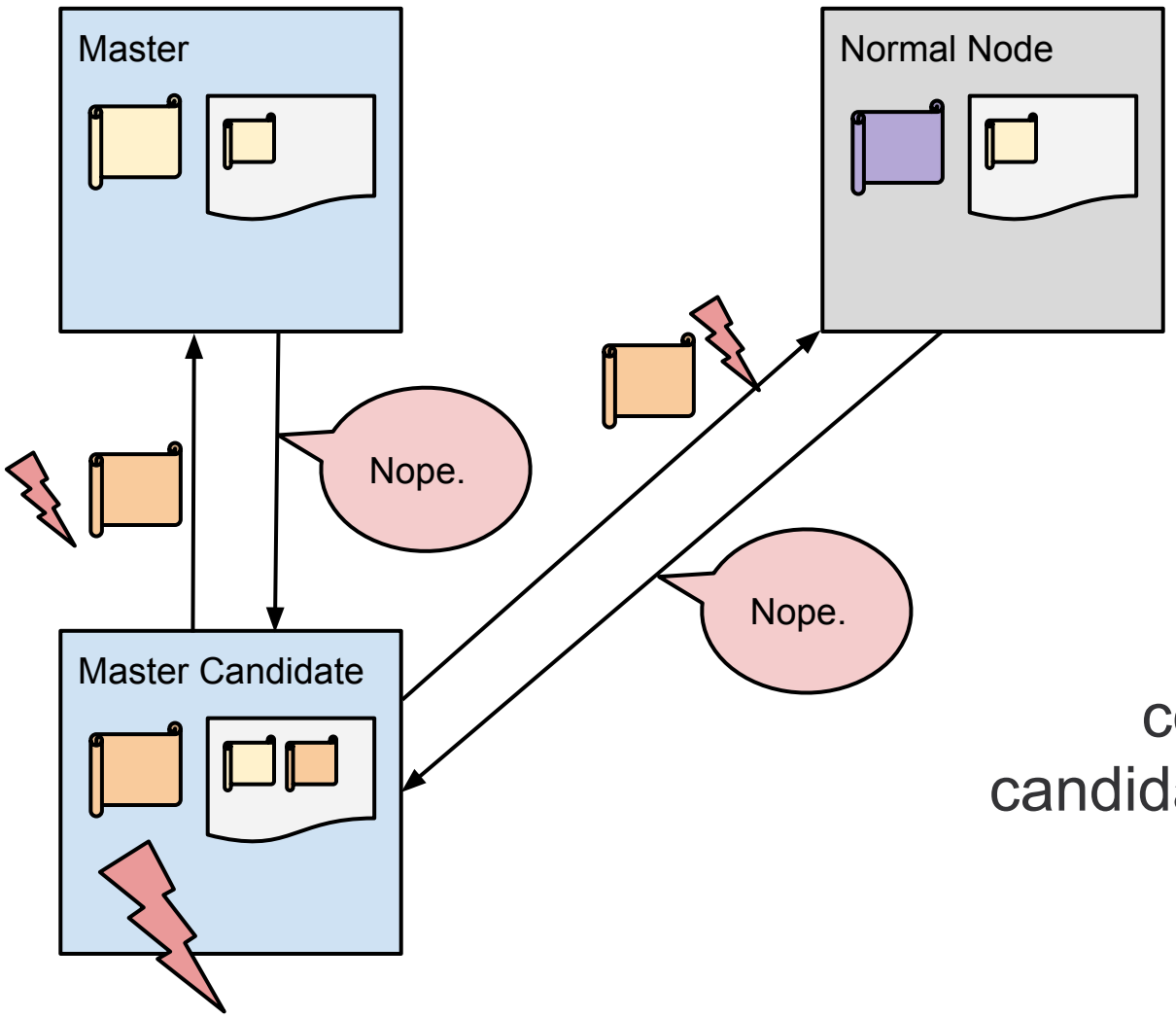
`authorized_keys`

Google Confidential and Proprietary



Attack Scenario: Master Candidates still do harm :(

Google Confidential and Proprietary



… but the key of a compromised master candidate can be removed independently.

Google Confidential and Proprietary

Comparison: Before and After

- Poor man's version of SSH usage
- One key pair for all nodes
- One compromised node can own the cluster

- Proper usage of SSH
- One key per node
- Only master candidate nodes' keys are put into the authorized_keys file
- A compromised normal node cannot spread the breakage
- An identified compromised master candidate's key can be removed from the cluster

Google Confidential and Proprietary

Implementation Challenges

- Don’t clutter up the ‘authorized_keys’ file
- Don’t lock yourself out
- Don’t add keys of random machines
- No private SSH key shall ever leave the node
- Don’t rely (too much) on SSL security
- gnt-cluster renew-crypto --new-ssh-keys
- Bootstrapping: from “Before” to “After”

Google Confidential and Proprietary

Can we do better?

Yes! No? Maybe?

Discuss this with us on Thursday in the Security Design session!

Google Confidential and Proprietary

Security Design Session

Ganeticon, Sep 2014

Helga Velroyen, helgav@google.com

Agenda

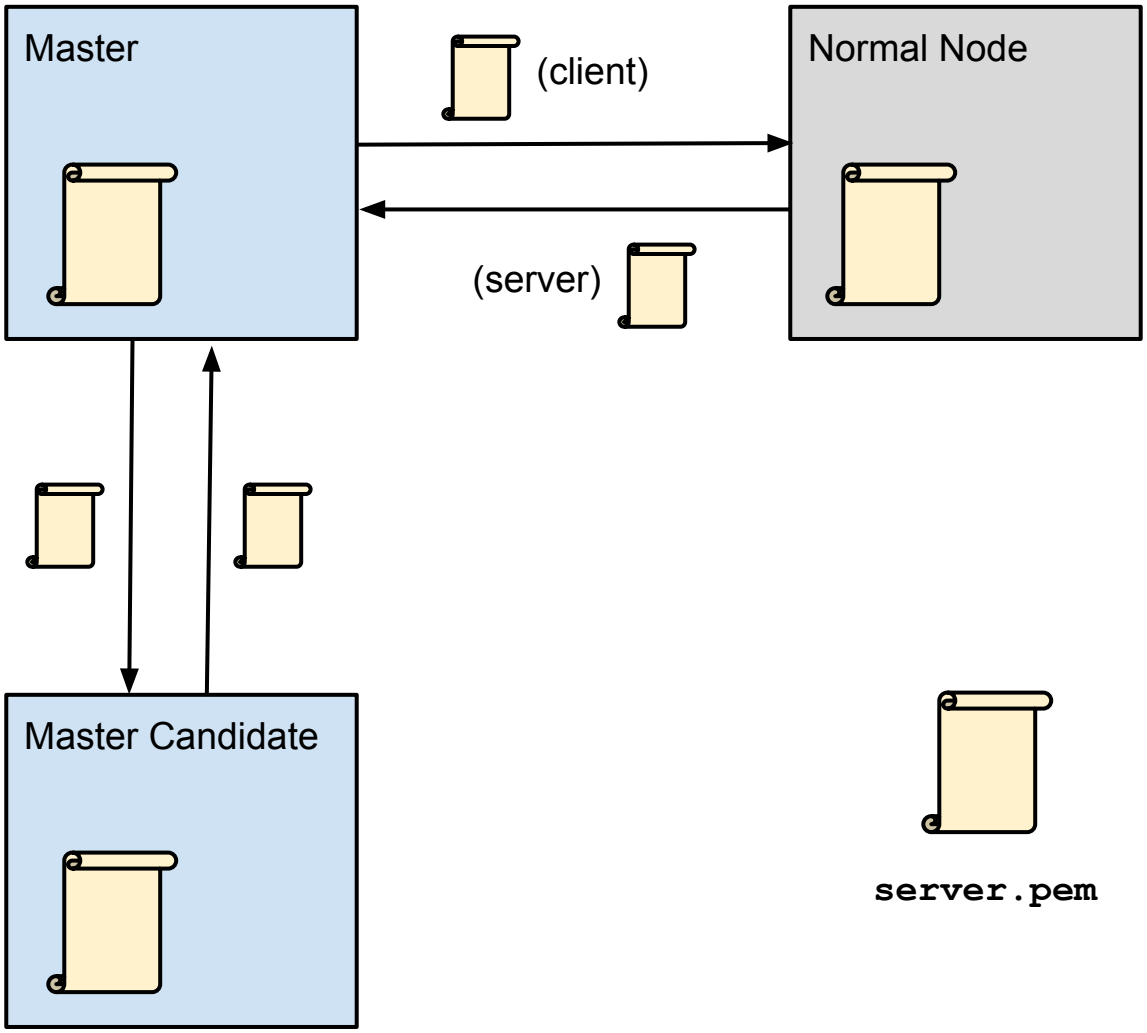
- SSL node security
- SSH node security
- Other security foo

Google Confidential and Proprietary

SSL

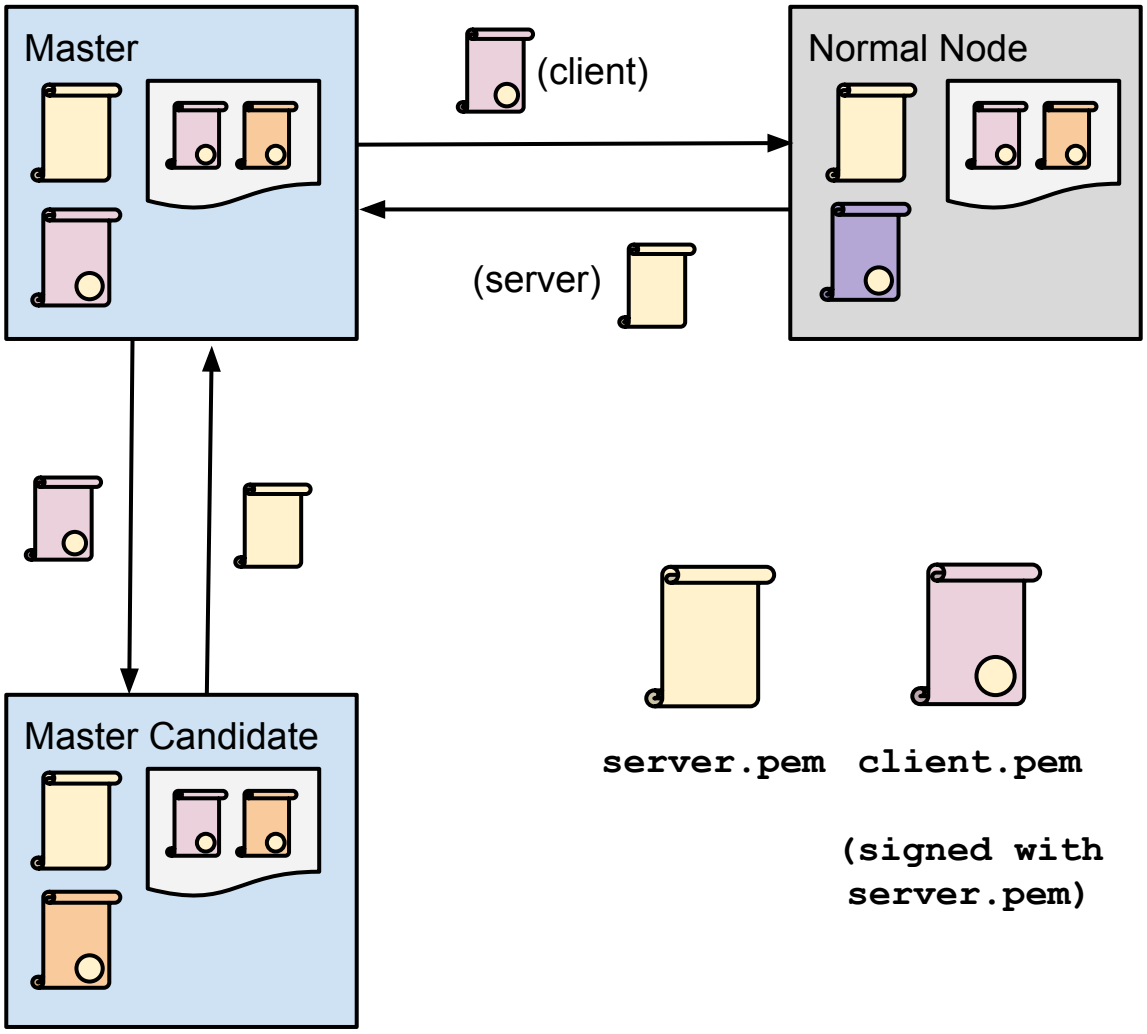
Node Security (Design Session)

RPC via SSL (pre 2.11)



Google Confidential and Proprietary

RPC via SSL Master Candidate Certificate Lists



Google Confidential and Proprietary

Pro Points of the Current Implementation

Pro:

- It’s better than before!
- It was implementable with reasonable effort.
- It meets the design goals.
- SSL key handling / creation / distribution is done by Ganeti completely without additional configuration effort necessary

Google Confidential and Proprietary

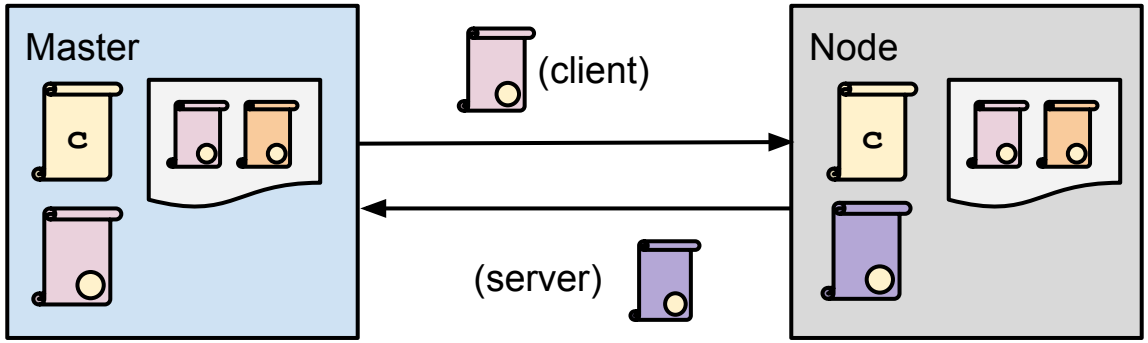
Pros and Cons of the Current Implementation

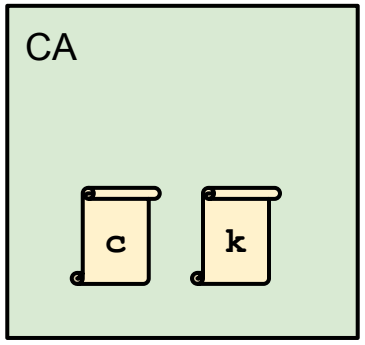
Con:

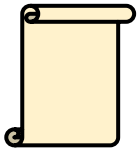
- Still not implemented as SSL is supposed to be used
- Still a shared certificate for all nodes (server.pem)
- Server.pem “abused” to sign client certificates
- Server certificate is self-signed
- No proper signing process (e.g. using signing requests)
- No real authority to sign client certificates (each node does it by themselves)
- No proper revocation process
- No integration in existing SSL infrastructure
- No hostname verification

Google Confidential and Proprietary

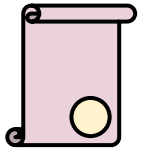
Use CA-signed certificates



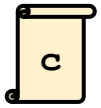




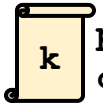
ca.pem



node.pem (signed by ca)



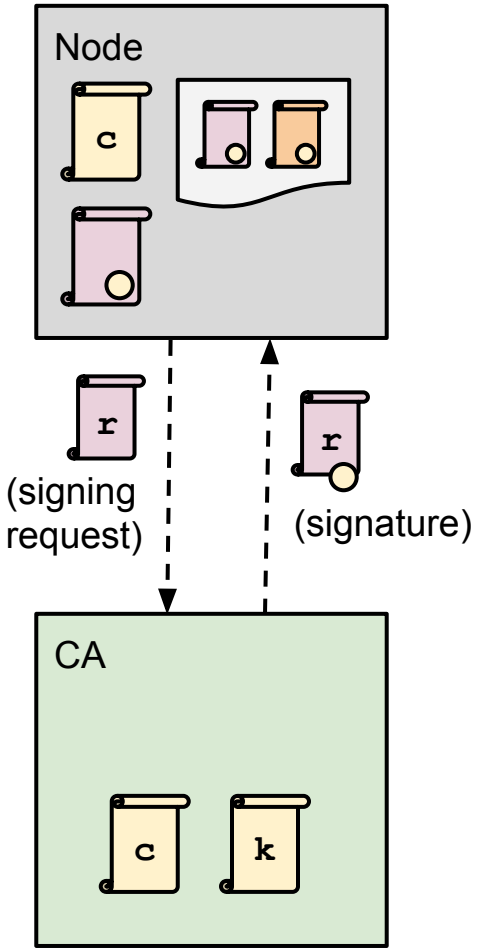
certificate

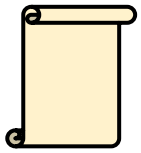


private key of certificate

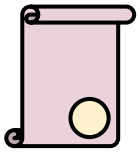
Google Confidential and Proprietary

Proper Signing Process by CA

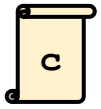




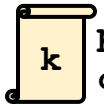
ca.pem



node.pem (signed by ca)



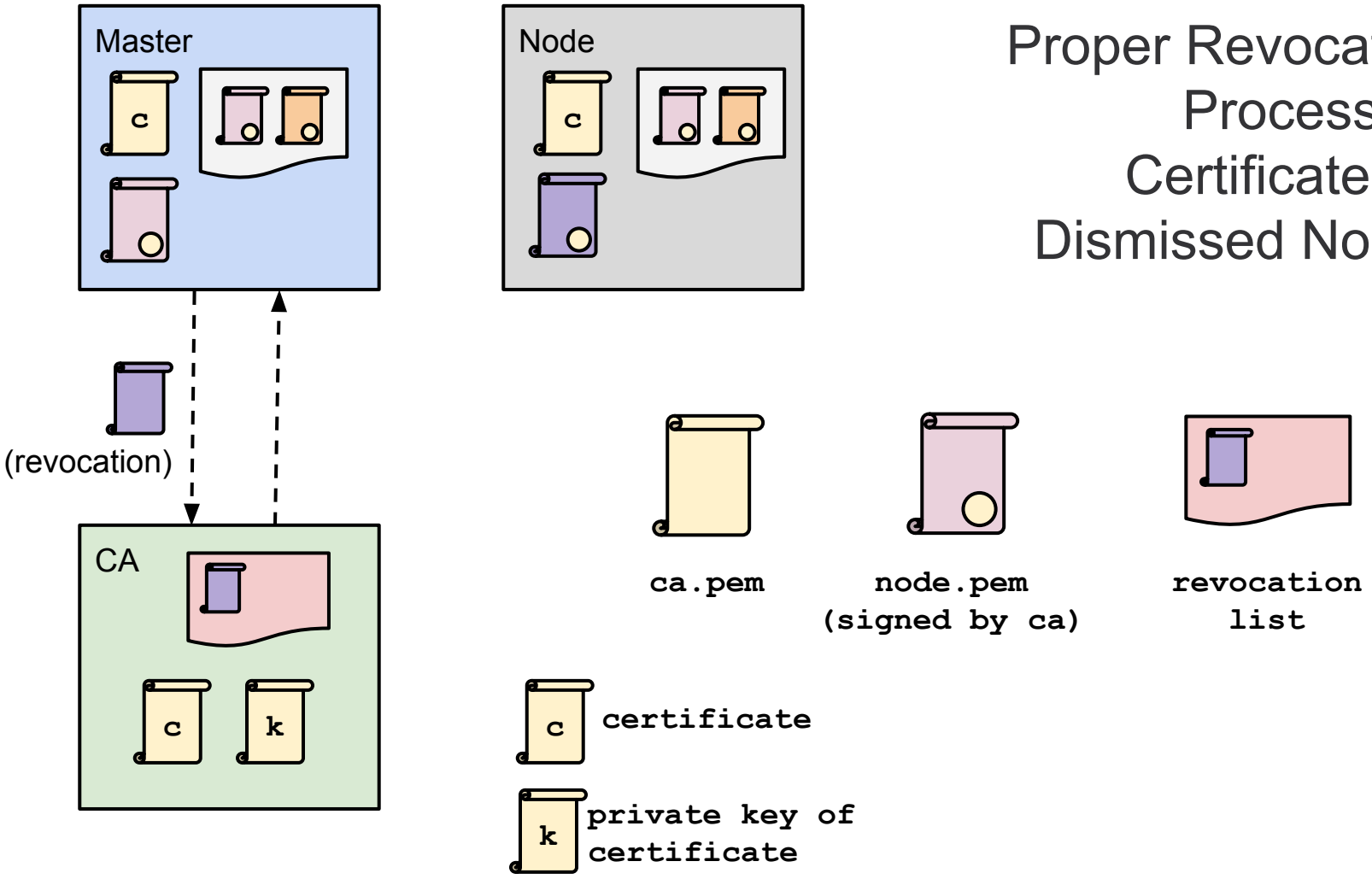
certificate



private key of certificate

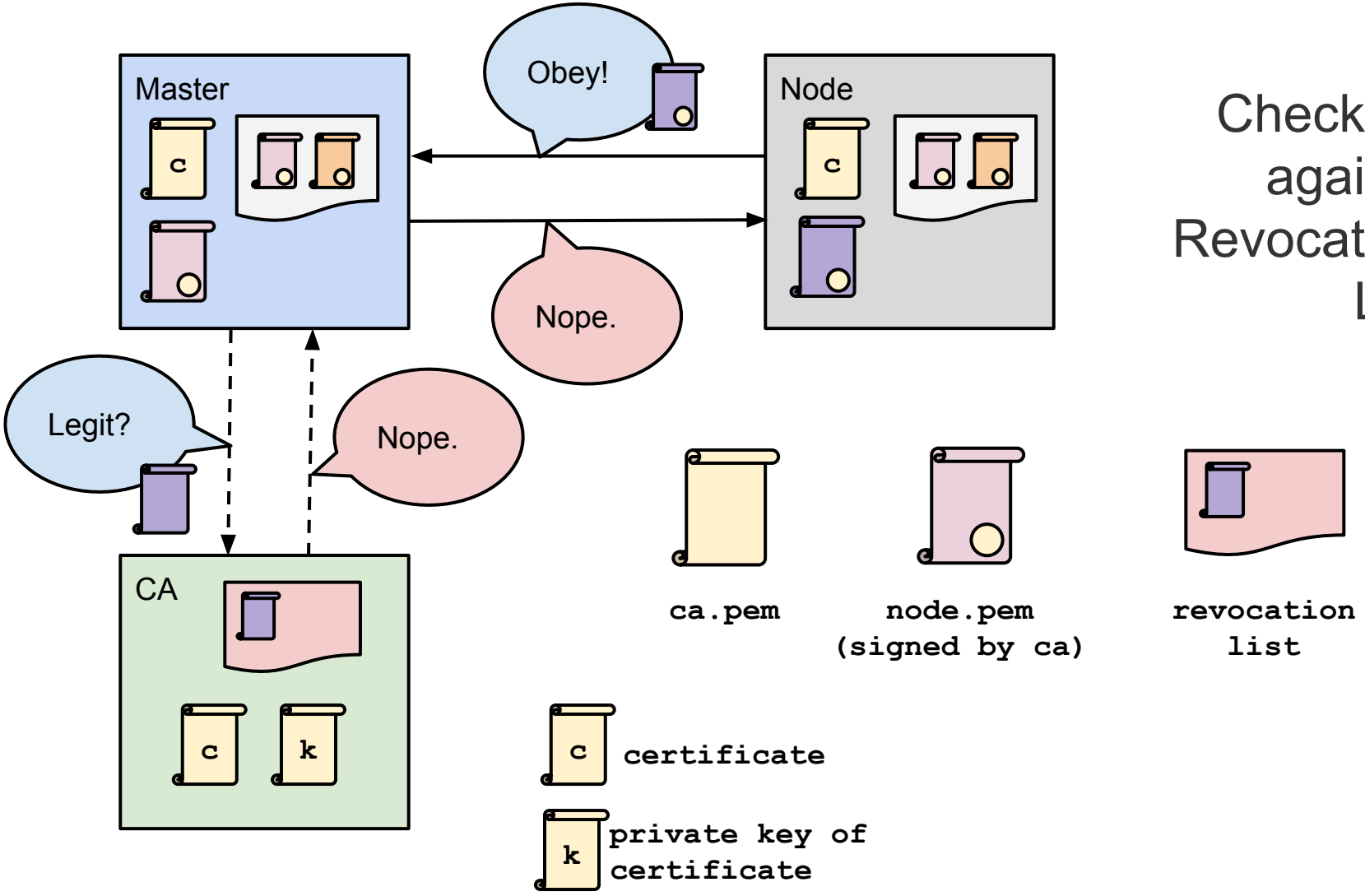
Google Confidential and Proprietary

Proper Revocation Process for Certificates of Dismissed Nodes



Google Confidential and Proprietary

Checking against Revocation List



Google Confidential and Proprietary

Pros of using a CA

- proper certificate signing process
- proper revocation (and checking against) process
- nodes only have public part of CA cert
- more like SSL was designed to be used
- CA certs itself can be signed properly (hierarchy)
- option to integrate existing / external CA infrastructure
- we could use proper hostname verification
- CA would be useful for imports / exports as well

Google Confidential and Proprietary

Internal vs. External CA

Internal, aka “Ganeti CA”:

- What nodes should be the CA?
  - {Master, MCs} = CA?
  - Dedicated set of Ganeti nodes?
- Lots of crap to be implemented!
- Should be easy to setup for Ganeti admins
- Should also work for small clusters

External CA:

- Larger organizations already run their CA infrastructure
- Other distributed systems have similar approach (puppet, bareos)
- Less logic to be implemented in Ganeti

Google Confidential and Proprietary

Your 2 cents?!

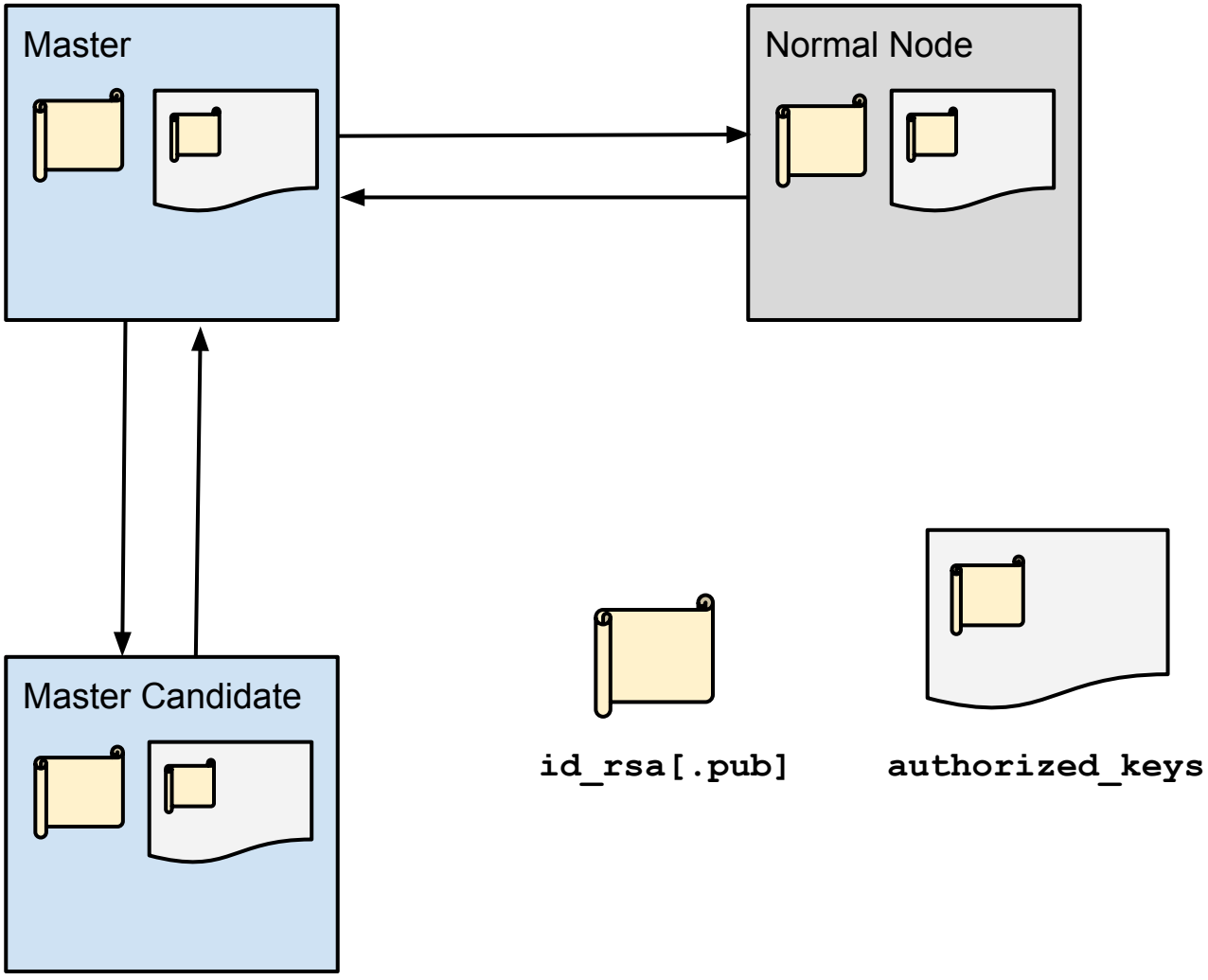
- What do you think of the current design? (server.pem + client.pem)
- What do you think of the proposal to use CAs?
- Is the missing CA a pain point for you so far?
- Would you prefer a Ganeti CA or an external CA?
- Any other SSL-related suggestions?

Google Confidential and Proprietary

SSH

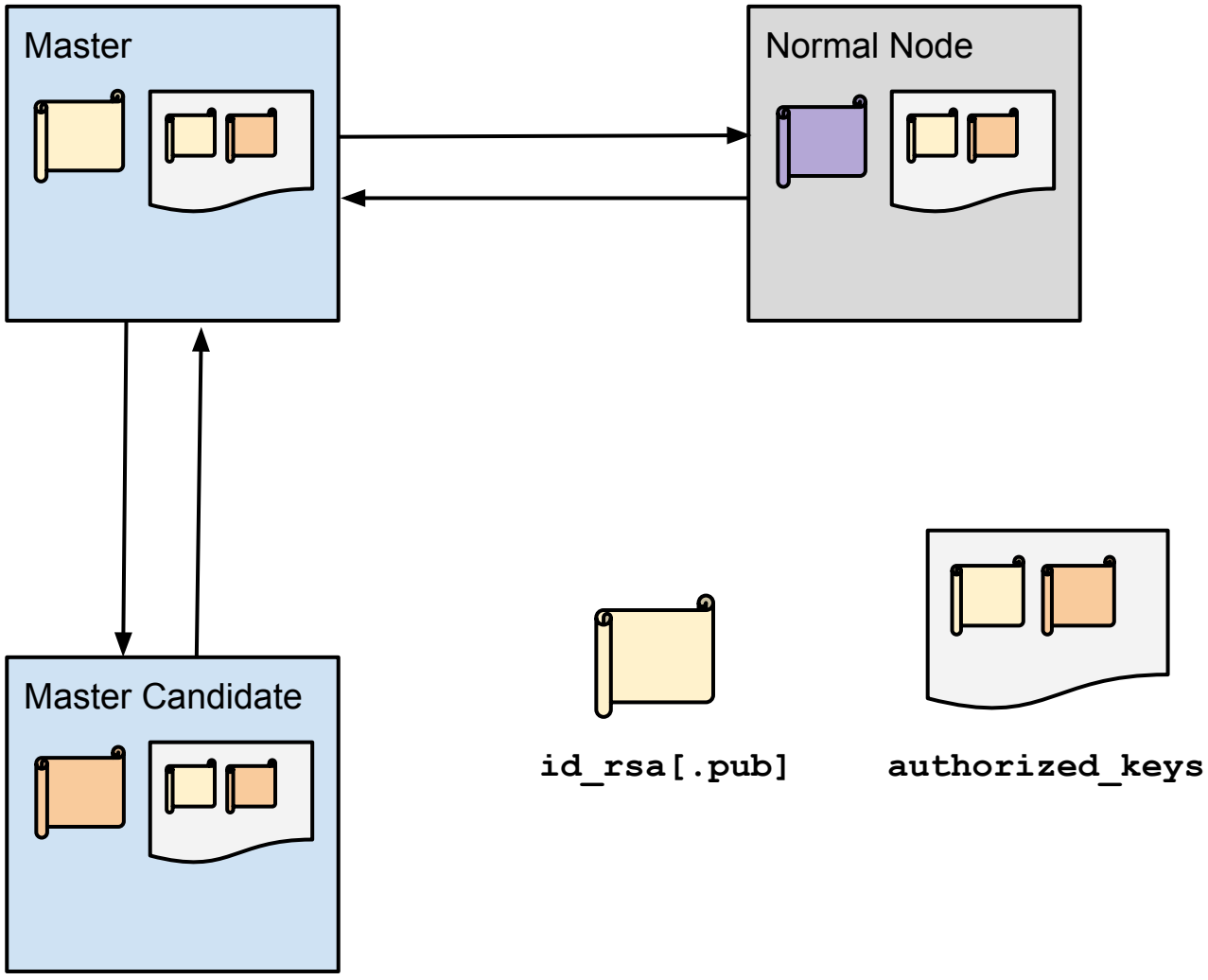
Node Security (Design Session)

SSH key setup (pre 2.13)



Google Confidential and Proprietary

SSH key setup in 2.13 (hopefully)



Google Confidential and Proprietary

Pros and Cons of the New Implementation

Pro:

- Increased security
- Compromised Normal Nodes can do no harm
- Compromised Master Candidates can be removed
- Implemented how SSH is supposed to be used

Con:

- Increased Complexity of Ganeti's SSH key handling
- No integration of external system so handle SSH key distribution
  - although doable by preventing Ganeti from handling the keys

Google Confidential and Proprietary

Your 2 cents?!

- Was the previous design a concern for you?
- Are you happy with the new design?
- Any further suggestions / improvements?

Google Confidential and Proprietary

Other Security Foo

Node Security (Design Session)

What else?!

- Anything else security related which we should talk about?

Google Confidential and Proprietary