RATS/EAT Verification Key Identification

Laurence Lundblade

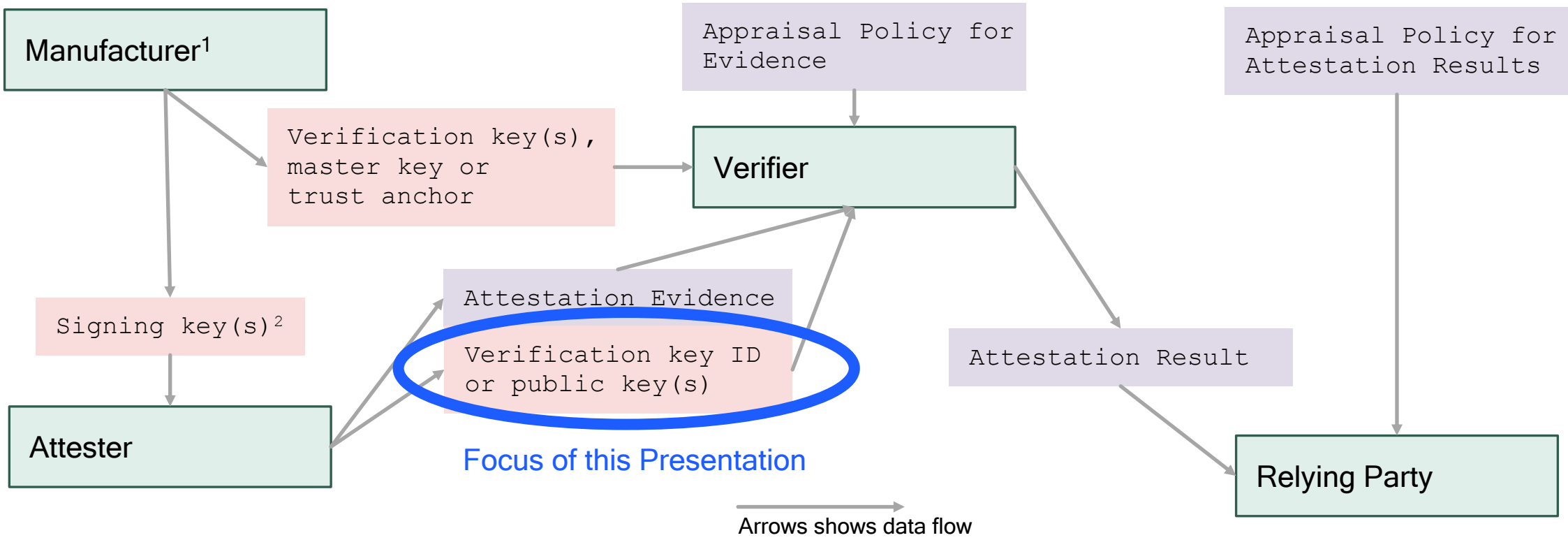
October 2020

Verifier Trust in the Attester

Cryptography (signing) is the only way an Attester can prove itself trustworthy to a Verifier

Manufacturer must set up key material such that:

- The Attester signs a nonce, evidence and such
- The Verifier verifies it



Notes:

1. The Manufacturer could be the actual device maker, an OS vendor, a trusted 3rd party, other or any combination of these.
2. There are several ways the signing keys may be established in the Attester. For example, a key pair may be generated in the Attester and the public key exported. This step is symbolic of the establishment of key material in the Attester with corresponding key material at the Manufacturer

Taxonomy of Verifier/Manufacturer Use of Keys

	Public key(s) Requires: authenticity	**Symmetric Key(s)** Requires authenticity, confidentiality	**Verification Service** Requires: authenticity
Infrequent Transfer	**TRUST_ANCHOR** • Trust Anchor sent • Per-device key is in Attestation Evidence • Trust Anchor used to verify per-device keys **PUB_KEY_DATABASE** • Manufacturer sends entire database of public keys • A million-key database for a million devices • Look up by key ID	**MASTER_SYMMETRIC_KEY** • Manufacturer sends a master symmetric key to Verifier • Verifier uses a KDF to derive individual device key **SYMMETRIC_KEY_DATABASE** • Manufacturer sends an entire database of symmetric keys to Verifier • A million-key database for a million devices • Look up by key ID	NOT POSSIBLE
Per-Verification	**PUB_KEY_LOOKUP** • Manufacturer maintains key database • Verifier sends key ID to Manufacturer • Manufacturer responds with public verification key for particular device	**SYMMETRIC_KEY_LOOKUP** • Manufacturer maintains key database • Verifier sends key ID to Manufacturer • Manufacturer responds with symmetric verification key for particular device	**VERIFICATION_SERVICE** • Keys are only at the Manufacturer • Verifier sends hash of to-be-verified bytes and key ID • Manufacturer responds with yeah/nay

Verification Key ID Taxonomy

Proposal is to describe this in the EAT document

	Description	Size efficiency	COSE/CWT/EAT	System Using
Key ID	• Byte string with no internal structure • Format of the key (COSE_KEY, DER-encoded…) is determined otherwise	Good – Small number of additional bytes	COSE kid header parameter	
URI	• Identifies where the verification key can be obtained via HTTP, HTTPS… • Content type of the data returned indicates the format of the key	Good - Small number of additional bytes	draft-ietf-cose-x509 when key is X.509. Other when it is not?	
Based on Claims	• Individual Claim or combination of Claims identifies the key. The UEID is a particular candidate. • The format of the key (COSE_KEY, DER-encoded…) is determined otherwise • Must decode payload before verification; makes decode/verification stack more complex; possible security issue of decoding unverified data	Best – No additional bytes	N/A (ID is in the Claims)	ARM PSA
Verification Key in Attestation Evidence (A Key, not a Key ID)	• For public keys only • Typically an X.509 certificate or equivalent such that the Verifier can chain it up to a trust anchor. May include intermediates between key and trust anchor.	Worst – X.509 certs can be large. Simpler schemes might just need a single public key.	draft-ietf-cose-x509	FIDO Android Attestation TPM

LL's Endorsement Assumption

- A mostly static document or file describing characteristics of one Attester or a class of Attesters
  - Could be 1 million Endorsements for 1 million devices
  - Could be 1 single Endorsement or 1 million devices
- Often contains a public key or key chain that can be used to verify trust in the Attester
  - Key could be for a single Attester for class of Attesters (e.g., a trust anchor)
  - Alternatively, a symmetric key for use with HMAC
    - Requires the endorsement be encrypted
- Often contains name-value attributes that describe characteristics of the Attester. Examples:
  - Implicit Claims like
    - Model and manufacturer
    - Level of security it offers: Hardware, TEE, Software…
    - Certifications
  - Reference values for comparison to Claims in Attestation Evidence
  - …
- Often is signed by an authority like the manufacturer of the Attester
- Example Endorsements:
  - X.509 Certificate with extensions for the Attester attributes
  - A new CBOR/COSE-based format

Add an Endorsement ID to EAT?

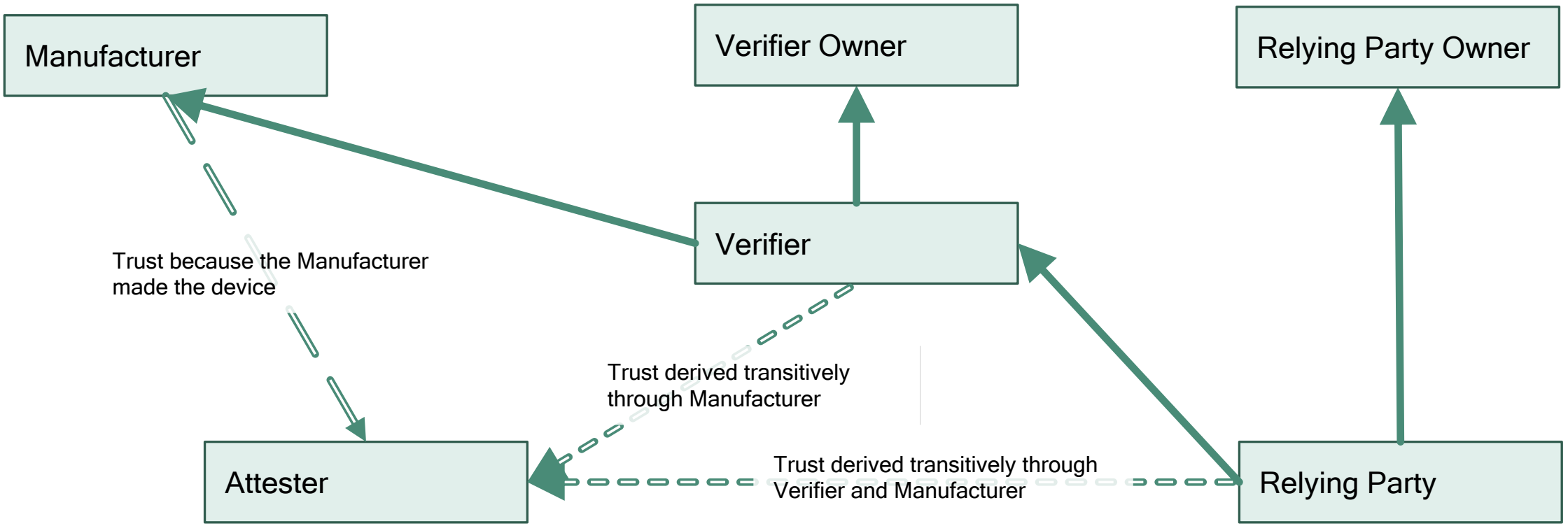
- COSE/CWT/EAT kid header parameter identifies only a key, **not** an Endorsement
- Propose adding an *Endorsement ID* and *Endorsement URL* to EAT
  - Replaces the EAT Origination claim (unused, poorly defined)
  - An alternative to key identification
    - Identifies the full Endorsement, not just the key
- Propose Endorsement ID and URL be COSE header parameters
  - Similar to `draft-ietf-cose-x509` and provides:
    - Option to include the full Endorsement and chain up to a root
    - Option to reference by a hash of the Endorsement
    - Option for a URL from which to fetch Endorsement
  - Endorsements rather than certificates
    - Verify knows to look inside for implicit claims
  - Allow use of X.509-format Endorsements AND other (future) Endorsement formats
    - A future format would be encrypted to be able to carry symmetric keys for verification

Extra Slides

Primary Trust Flow

The primary point of Attestation is for the Relying Party to trust the Attester / Manufacturer

If trust on any of these links fails, attestation fails and the Relying Party doesn't get what it is promised



Secondary Trust Flow

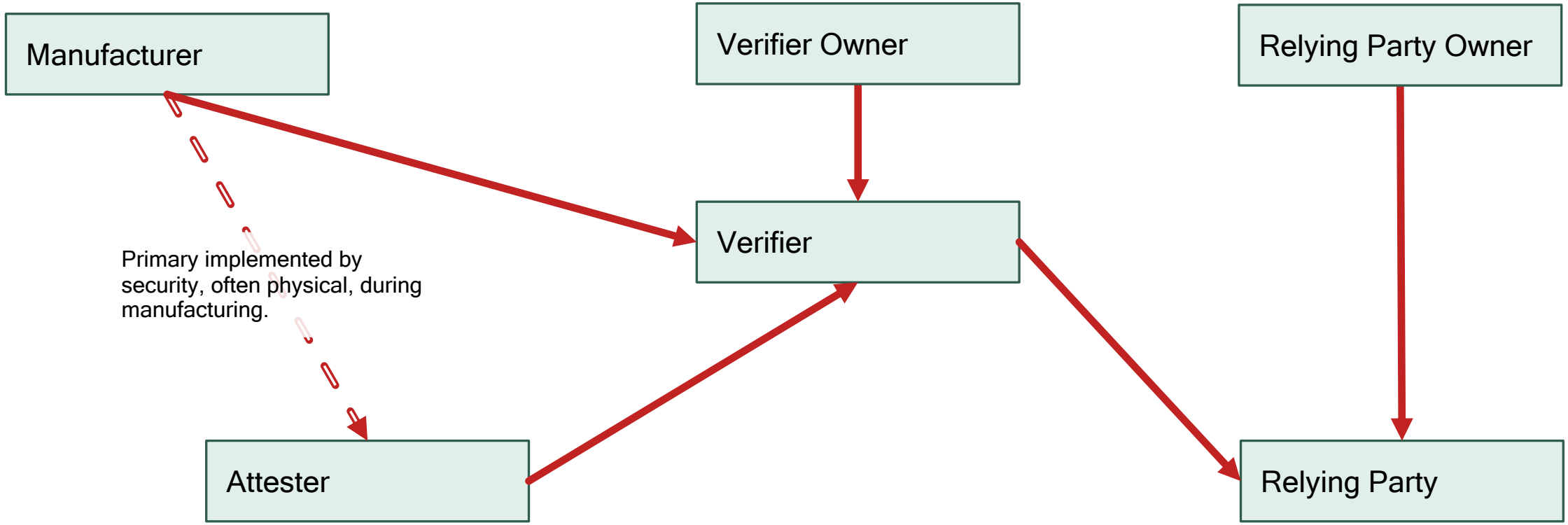
Trusted ← Trusting

Is about confidentiality and PII

If these links fail, the attacker gets PII about the user or information useful to attack the device

Authenticate target of the arrow and encrypting

In simple scenarios where there is no PII and no information useful for attack, this is optional



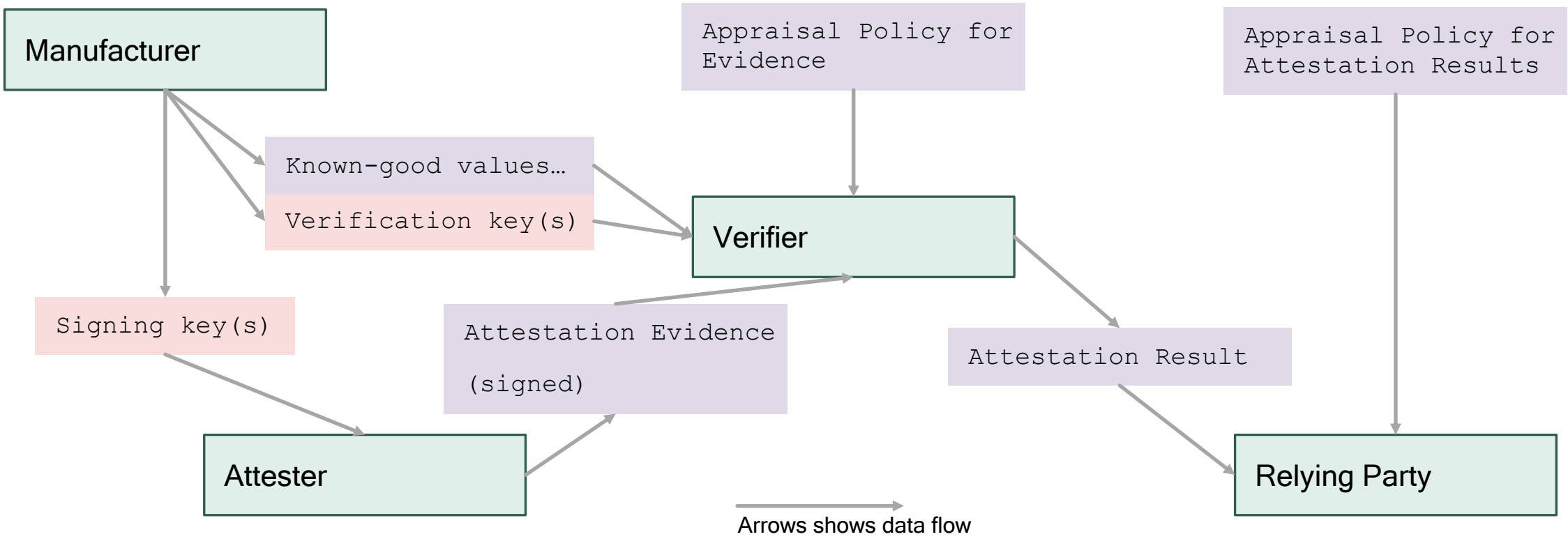
Verifier Trust in the Attester

Cryptography (signing) is the only way an Attester can prove itself trustworthy to a Verifier

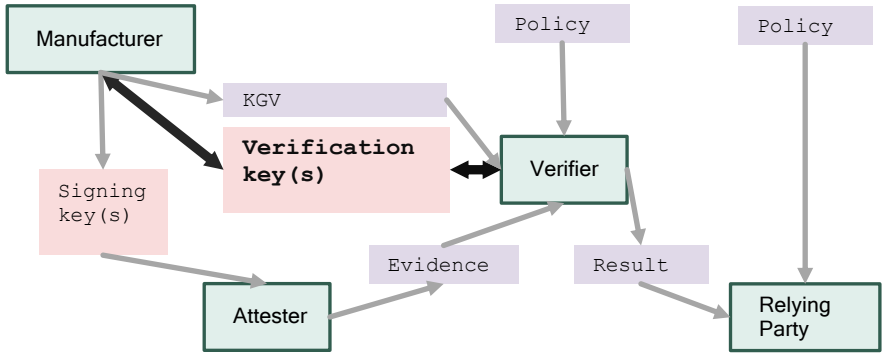
Manufacturer must set up key material such that:

- The Attester signs a nonce, evidence and such
- The Verifier verifies it

Note: The Manufacturer could be the actual device maker, an OS vendor, a trusted 3rd party, other or any combination of these.



Taxonomy of Verifier/Manufacturer Use of Keys



	Public key(s)	Symmetric Key(s)	Verification Service
Infrequent Transfer			
Per-Verification			

- Infrequent transfer
  - One-time (e.g. root of trust)
  - Batches
  - Quarterly / annually
- Per-verification
  - Verifier interacts with Manufacturer on ***every*** verification

- Public-key cryptography
  - Private key in Attester; public key in Verifier
  - Example: ECDSA with COSE_Sign1
  - Public key(s) transferred from Manufacturer to Verifier
- Symmetric cryptography
  - Attester and Verifier have the identical same key
  - Key must be secret in both the Verifier and Attester!
  - Example HMAC with COSE_Mac0
- Verification service
  - Crypto and key type doesn't matter. They are internal to the Manufacturer.
  - Manufacturer sends hash of to-be-signed or to-be-MACed bytes and key ID; receives yeah/nay