TBBR CoT using FCONF

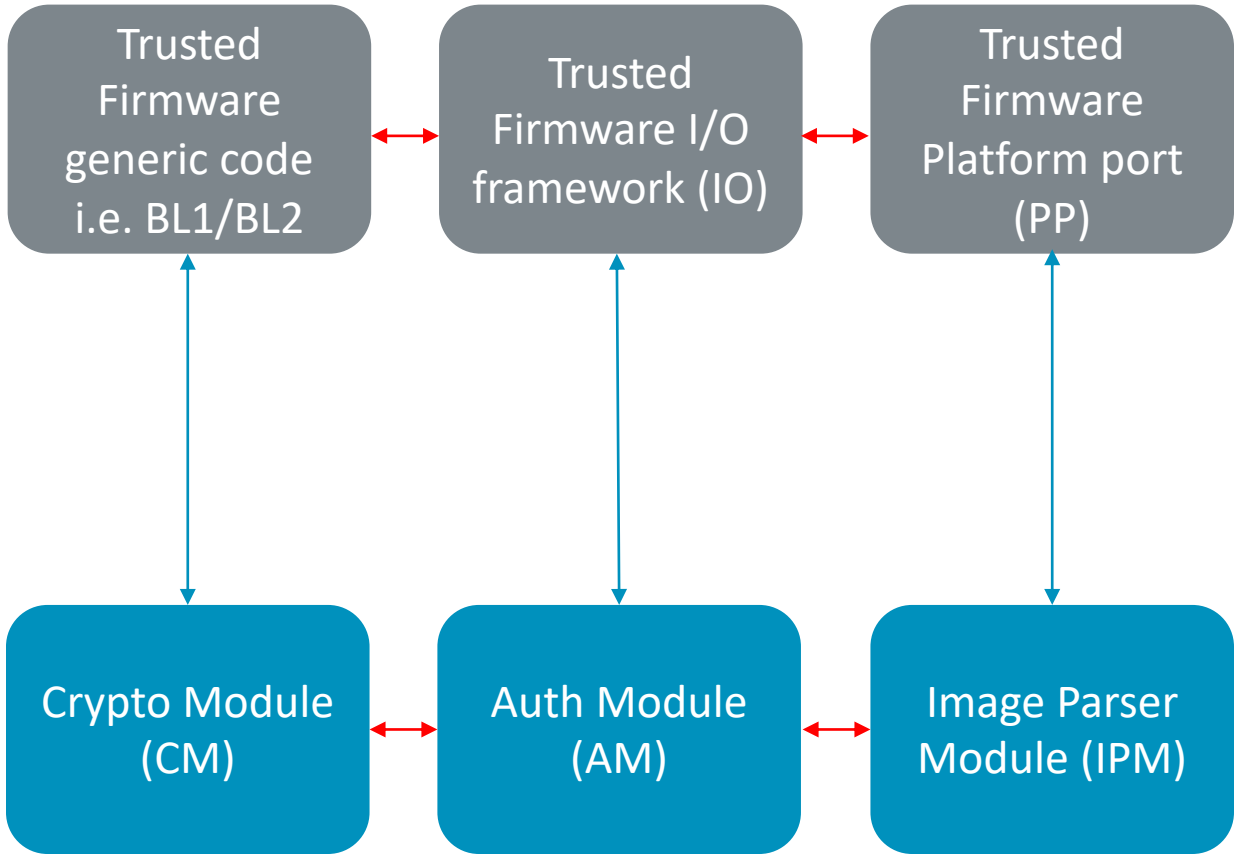
Manish Badarkhe
21/05/2020

Agenda

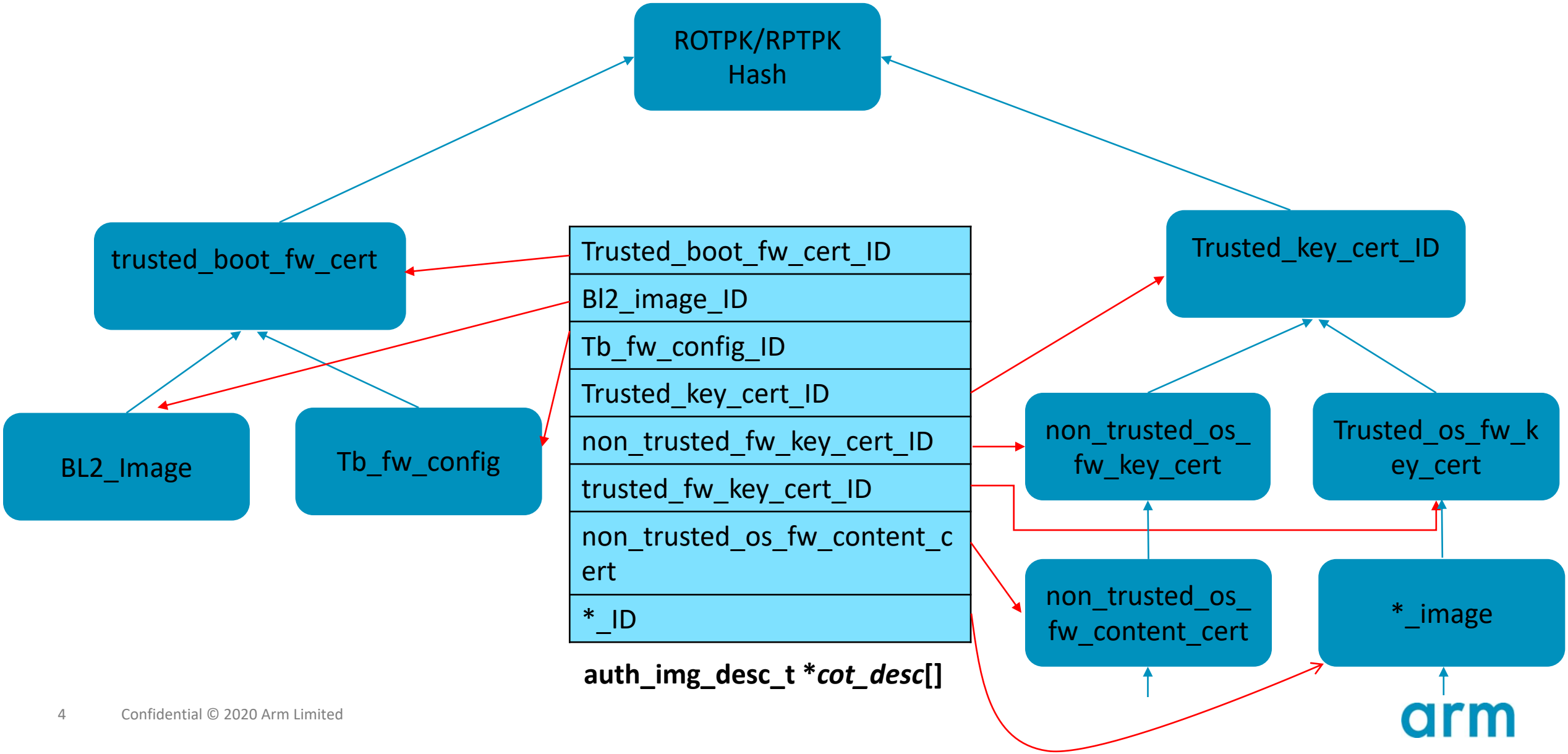
- Basic overview of authentication framework
- Current design of TBBR CoT descriptors
- Use FCONF for CoT descriptors

Authentication Framework

- GEN and I/O
  - load images using I/O framework
  - Authenticate images using AM
- Platform port
  - Specify CoT information for each image
  - Provision to get ROTPK or hash of it
- Modules
  - Verify CoT by using APIs exported by CM, AM and IPM

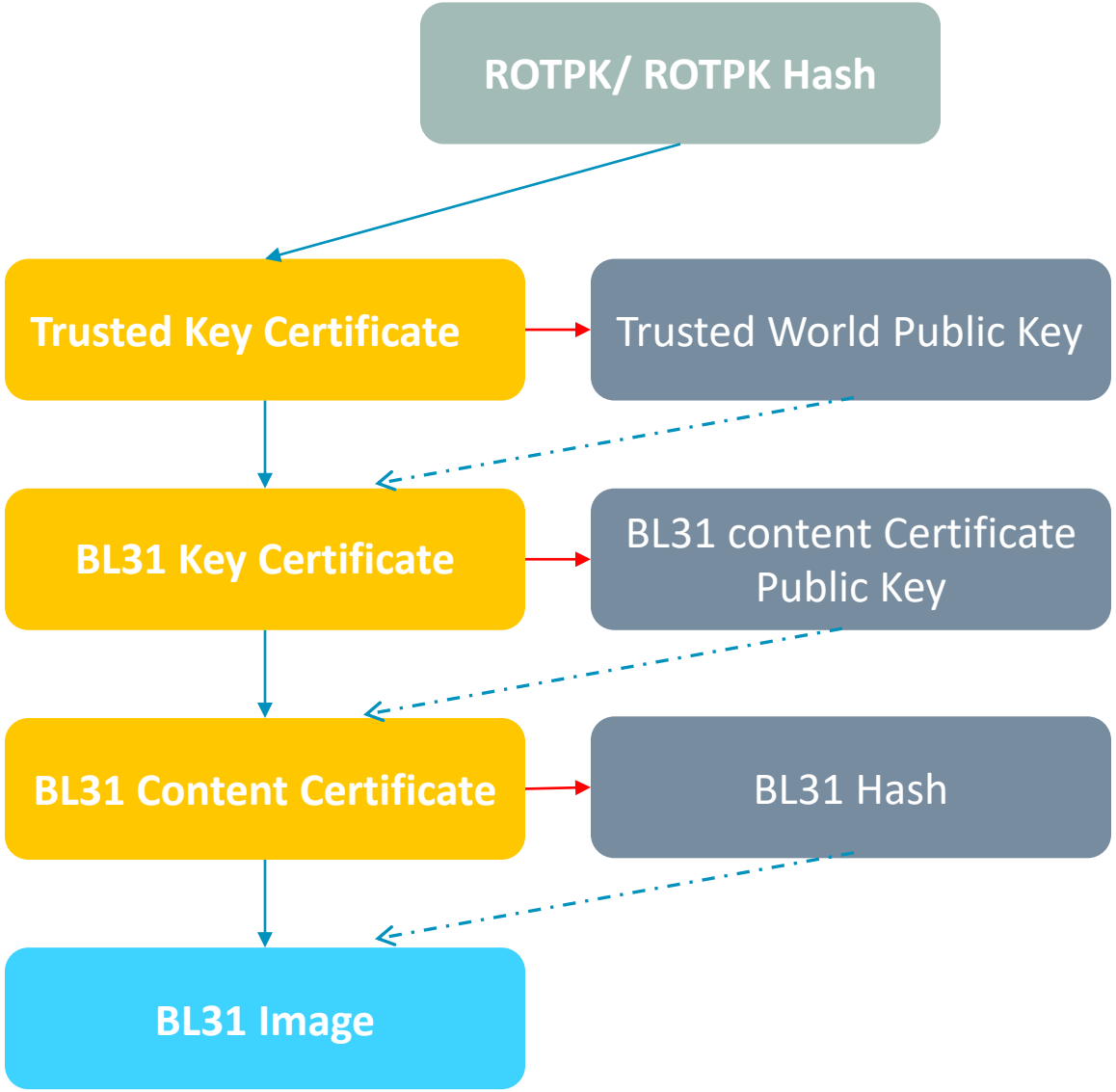


Chain of Trust 1/2



Chain of Trust 2/2

- Authenticate raw image using appropriate certificates
- Example shows how BL31 image gets authenticated using CoT.
- BL31 has to go through all necessary certificate authentication before getting authenticated.



Current Implementation of CoT Descriptor

Certificate Image Descriptor

```
/* trusted_boot_fw_cert */
const auth_img_desc_t trusted_boot_fw_cert = {
        .img_id = TRUSTED_BOOT_FW_CERT_ID,
        .img_type = IMG_CERT,
        .parent = NULL,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &subject_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_PARAMS]) {
                [0] = {
                        .type_desc = &tb_fw_hash,
                        .data = {
                                .ptr = (void *)tb_fw_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                },
                [1] = {
                        .type_desc = &tb_fw_config_hash,
                        .data = {
                                .ptr = (void *)tb_fw_config_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                },
                [2] = {
                        .type_desc = &hw_config_hash,
                        .data = {
                                .ptr = (void *)hw_config_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                }
        }
};
```

Raw Image Descriptor

```
/* HW Config */
const auth_img_desc_t hw_config = {
        .img_id = HW_CONFIG_ID,
        .img_type = IMG_RAW,
        .parent = &trusted_boot_fw_cert,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_HASH,
                        .param.hash = {
                                .data = &raw_data,
                                .hash = &hw_config_hash
                        }
                }
        }
};
```

Authentication Parameter

```
auth_param_type_desc_t hw_config_hash = AUTH_PARAM_TYPE_DESC(
        AUTH_PARAM_HASH, HW_CONFIG_HASH_OID);
```

CoT for BL31 Image 1/2

```
static const auth_img_desc_t soc_fw_content_cert = {
        .img_id = SOC_FW_CONTENT_CERT_ID,
        .img_type = IMG_CERT,
        .parent = &soc_fw_key_cert,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &soc_fw_content_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_PARAMS]) {
                [0] = {
                        .type_desc = &soc_fw_hash,
                        .data = {
                                .ptr = (void *)soc_fw_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                },
                [1] = {
                        .type_desc = &soc_fw_config_hash,
                        .data = {
                                .ptr = (void *)soc_fw_config_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                }
        }
};
```

```
static const auth_img_desc_t bl31_image = {
        .img_id = BL31_IMAGE_ID,
        .img_type = IMG_RAW,
        .parent = &soc_fw_content_cert,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_HASH,
                        .param.hash = {
                                .data = &raw_data,
                                .hash = &soc_fw_hash
                        }
                }
        }
};
```

CoT for BL31 Image 2/2

```
static const auth_img_desc_t trusted_key_cert = {
        .img_id = TRUSTED_KEY_CERT_ID,
        .img_type = IMG_CERT,
        .parent = NULL,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &subject_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_PARAMS]) {
                [0] = {
                        .type_desc = &trusted_world_pk,
                        .data = {
                                .ptr = (void *)trusted_world_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                },
                [1] = {
                        .type_desc = &non_trusted_world_pk,
                        .data = {
                                .ptr = (void *)non_trusted_world_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                }
        }
};
```

```
static const auth_img_desc_t soc_fw_key_cert = {
        .img_id = SOC_FW_KEY_CERT_ID,
        .img_type = IMG_CERT,
        .parent = &trusted_key_cert,
        .img_auth_methods = (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &trusted_world_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_PARAMS]) {
                [0] = {
                        .type_desc = &soc_fw_content_pk,
                        .data = {
                                .ptr = (void *)content_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                }
        }
};
```

Image descriptor

Each image is represented using descriptor. Descriptor mainly consist of below data

- img_id:
  - Image Id of either certificate or raw-image
- parent:
  - Parent image of this descriptor
- img_type:
  - Either Raw image or Cert image
- img_auth_methods:
  - Currently supported 3 auth methods
    - HASH
    - SIG
    - NV Counter (Antiroll-back)
- authenticated_data:
  - Used to store public key and hash data after authentication.

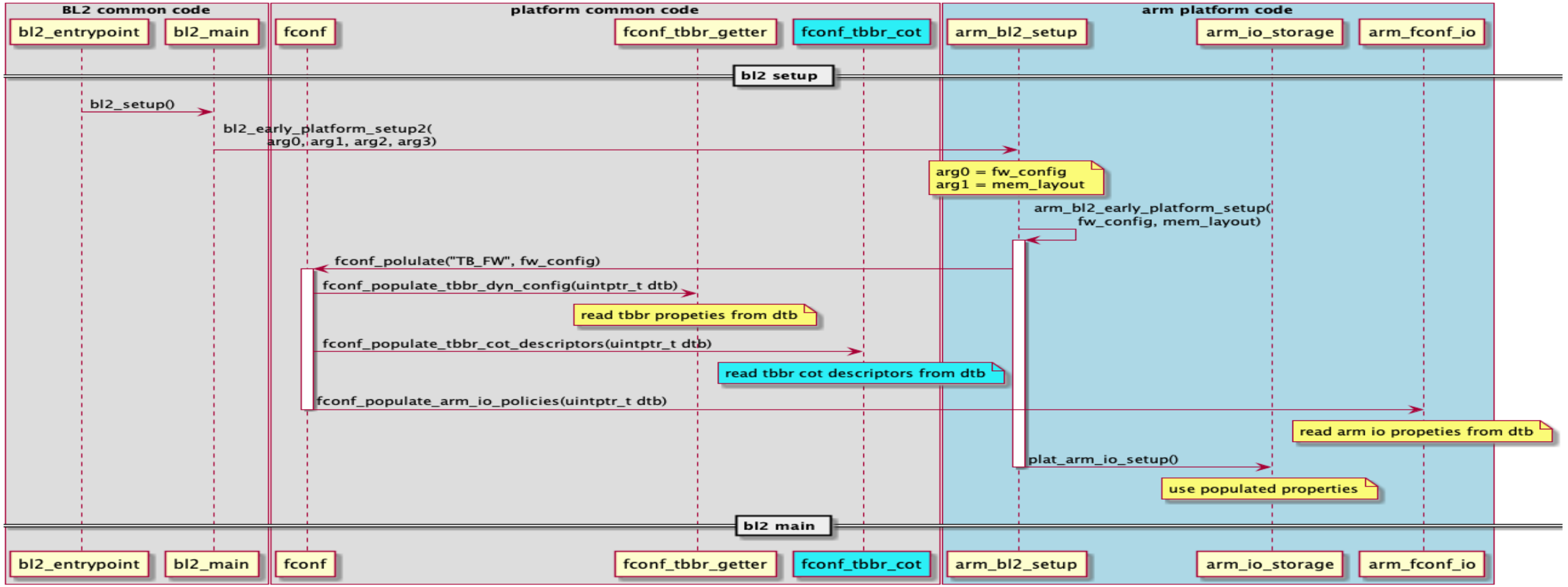
Sample certificate and image descriptor in device tree

```
Certificates {
        compatible =  arm,tbbr-cert-cot-descriptors”
        trusted-boot-fw-cert {
                /* default: false use only
                    if this is root node */
                root = <true>;
                phandle = <TRUSTED_BOOT_FW_CERT_ID>;
                /* default: NULL use only
                   if parent is not NULL */
                parent = <&cert_desc>;
                /* default: NULL use this auth method
                    and cookie in parent is not  NULL */
                auth-method = ”sign”;
                /* OID to extract key from parent image */
                auth-cookie = “sig-cookie“;

                antirollback-cookie = “trusted-nvcounter”;
         };
         Next-cert {
         };
}
```

```
raw-Images {
        compatible =  arm, tbbr-img-cot-descriptors"
        hw-config  {
                phandle = <HW_CONFIG_ID>;
                parent = < TRUSTED_BOOT_FW_CERT_ID>;
                auth-method = “hash”;
                auth-cookie =HW_CONFIG_HASH_OID;
        };
        next-image {

        };
}
```


arm

Use FCONF for TBBR COT



FCONF_REGISTER_POPULATOR(TB_FW, tbbr_cot, fconf_populate_tbbr_cot_descriptors);

Conclusion

- Pros
  - Add descriptors seamlessly with proper CoT.
  - Helps to add descriptors in more readable format
  - Avoids using static initialized buffer and descriptors in the code.

- Cons
  - There may be penalty of memory as need to allocate memory statically for MAX_NUMBER of descriptors in any case.

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה