
punchboot

Release 1.0.0

Jonas Blixt

Feb 18, 2024

CONTENTS

1	Introduction	1
1.1	License	1
1.2	Distinguishing Features	1
2	Getting started	3
3	Design	5
4	Boot Process	7
5	API reference	9
5.1	bio — Block device API	9
5.2	crypto — Crypto API	9
5.3	plat — Platform API	12
6	Indices and tables	15
	Python Module Index	17
	Index	19

INTRODUCTION

Punchboot is a bootloader for embedded systems. It has a strong emphasis on security and compactness.

The following architectures are supported:

- Armv7a
- Armv8a

The following SoC's/platform's are supported:

- nxp imx6ul
- nxp imx8x
- nxp imx8m
- qemu virt

1.1 License

Punchboot is premissively licensed under the BSD 3 licens.

1.2 Distinguishing Features

Only supports signed payloads

Punchboot only supports signed payloads which reduces the risk for configuration errors and reduces the logic in the boot code.

Highly configurable

Most features can be enabled or disabled through the Kconfig interface

Host tooling

Punchboot-tools provides a set of tools and libraries for interacting with the bootloader

- Punchboot CLI and a C library
- Python wrapper

These can easily be extended to support board/production specific commands.

Optimized for speed

Most drivers, in particular block device drivers and hashing accelerators use DMA to maximize transfer speed.

GETTING STARTED

CHAPTER
THREE

DESIGN

BOOT PROCESS

This document describes the main boot flow. The boot process is modular and can support many different use cases.

API REFERENCE

5.1 bio — Block device API

Block device layer

Source code: `include/pb/bio.h`, `src/bio.c`

Warning: doxygenfile: Cannot find file “include/drivers/block/bio.h”

5.2 crypto — Crypto API

Crypto API

Source code: `include/drivers/pb/crypto.h`, `src/crypto.c`

Punch BOOT

Copyright (C) 2023 Jonas Blixt jonpe960@gmail.com

SPDX-License-Identifier: BSD-3-Clause

Defines

CRYPTO_MD_MAX_SZ

Largest message digest in bytes

HASH_MD5

HASH_MD5_BROKEN

`HASH_SHA256`

`HASH_SHA384`

`HASH_SHA512`

`DSA_EC_SECP256r1`

`DSA_EC_SECP384r1`

`DSA_EC_SECP521r1`

Typedefs

`typedef uint32_t hash_t`

`typedef uint32_t dsa_t`

`typedef uint32_t key_id_t`

Functions

`int hash_init(hash_t alg)`

Initialize the hashing context. The crypto API only supports one running context, calling this function will reset the context.

param[in] alg Hashing algorithm to use

Returns

PB_OK on success, -PB_ERR_PARAM, on invalid hash alg

`int hash_update(const void *buf, size_t length)`

Update currently running hash context with data

Parameters

- **buf** – [in] Input buffer to hash
- **length** – [in] Length of buffer

Returns

PB_OK on success

`int hash_update_async(const void *buf, size_t length)`

Update current running hash context with data. This function might be implemented by drivers for hardware accelerated hashing functions. Typically it will enqueue DMA descriptors and not wait for completion.

The underlying driver should check if there is a job in progress and block before enqueueing additional descriptors.

Parameters

- **buf** – [in] Input buffer to hash
- **length** – [in] Length of buffer

Returns

PB_OK on success

int **hash_copy_update**(const void *src, void *dest, size_t length)

Some hardware/drivers support both updating the hash context and copy the input buffer to another memory destination.

If the underlying driver does not implement the copy_update API the crypto module will use memcpy.

Parameters

- **src** – [in] Input/Source buffer to hash/copy
- **dest** – [in] Destination address
- **length** – [in] Length of input buffer

Returns

PB_OK on success

int **hash_final**(uint8_t *digest_output, size_t length)

Finalize hashing context.

This function will block if there is an async job queued.

Parameters

- **digest_output** – [out] Message digest output buffer
- **length** – [in] Length of output buffer

Returns

PB_OK on success

int **hash_add_ops**(const struct *hash_ops* *ops)

Register hash op's

Used by drivers to expose hashing functions.

Parameters

ops – [in] Hashing op's structure

Returns

PB_OK on success

int **dsa_verify**(*dsa_t* alg, const uint8_t *der_signature, size_t signature_length, const uint8_t *der_key, size_t key_length, *hash_t* md_alg, uint8_t *md, size_t md_length, bool *verified)

int **dsa_add_ops**(const struct *dsa_ops* *ops)

void **hash_print**(const char *prefix, uint8_t *digest, size_t length)

struct **hash_ops**

Public Members

const char ***name**

Name of hash op's provider

uint32_t **alg_bits**

Bit field that indicates supported algs

int (***init**)(*hash_t* alg)

Hash init call back

int (***update**)(const void *buf, size_t length)

Hash update callback

int (***update_async**)(const void *buf, size_t length)

Optional asynchronous update callback. The implementation is expected to queue/prepare an hash update and block if it's called again, until the current operation is completed

int (***copy_update**)(const void *src, void *dest, size_t length)

Optional copy and update. This function will simultaiously copy and hash data

int (***final**)(uint8_t *digest_out, size_t length)

Finalize and output message digest

struct **dsa_ops**

Public Members

const char ***name**

uint32_t **alg_bits**

int (***verify**)(const uint8_t *der_signature, size_t signature_length, const uint8_t *der_key, size_t key_length, *hash_t* md_alg, uint8_t *md, size_t md_length, bool *verified)

5.3 plat — Platform API

The platform API constitutes the minimum and mandatory functions each platform must provide.

The 'plat_init' function in each platform is expected to perform the following basic initialization:

- Initialize a watchdog
- Configure the systick
- Load and decode system boot reason

- Initialize debug console
- Configure the MMU

The ‘plat_init’ and ‘plat_board_init’ functions are called early during boot, se *Boot Process* for more details.

Source code: [include/pb/plat.h](#)

Punch BOOT

Copyright (C) 2020 Jonas Blixt jonpe960@gmail.com

SPDX-License-Identifier: BSD-3-Clause

Functions

int **plat_init**(void)

Initialize the platform

Returns

PB_OK on success or a negative number

int **plat_board_init**(void)

Initialize the board

Returns

PB_OK on success or a negative number

void **plat_reset**(void)

Platform specific reset function

unsigned int **plat_get_us_tick**(void)

Read platform/system tick

Returns

current micro second tick

void **plat_wdog_kick**(void)

Kick watchdog

int **plat_boot_reason**(void)

Returns a platform specific boot reason as an integer.

Returns

>= 0 for valid boot reasons or a negative number on error

const char ***plat_boot_reason_str**(void)

Returns a platform specific boot reason as a string

Returns

Boot reason string or ""

int **plat_get_unique_id**(uint8_t *output, size_t *length)

Platform / SoC Unique data. This function is called by the device_uuid module to generate a device unique identifier

Parameters

- **output** – [out] Unique data output
- **length** – [inout] Size of output buffer and result length

Returns

PB_OK on success or a negative number

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

b

bio, 9

c

crypto, 9

p

plat, 12

B

bio
 module, 9

C

crypto
 module, 9
 CRYPTO_MD_MAX_SZ (C macro), 9

D

dsa_add_ops (C++ function), 11
 DSA_EC_SECP256r1 (C macro), 10
 DSA_EC_SECP384r1 (C macro), 10
 DSA_EC_SECP521r1 (C macro), 10
 dsa_ops (C++ struct), 12
 dsa_ops::alg_bits (C++ member), 12
 dsa_ops::name (C++ member), 12
 dsa_ops::verify (C++ member), 12
 dsa_t (C++ type), 10
 dsa_verify (C++ function), 11

H

hash_add_ops (C++ function), 11
 hash_copy_update (C++ function), 11
 hash_final (C++ function), 11
 hash_init (C++ function), 10
 HASH_MD5 (C macro), 9
 HASH_MD5_BROKEN (C macro), 9
 hash_ops (C++ struct), 11
 hash_ops::alg_bits (C++ member), 12
 hash_ops::copy_update (C++ member), 12
 hash_ops::final (C++ member), 12
 hash_ops::init (C++ member), 12
 hash_ops::name (C++ member), 12
 hash_ops::update (C++ member), 12
 hash_ops::update_async (C++ member), 12
 hash_print (C++ function), 11
 HASH_SHA256 (C macro), 9
 HASH_SHA384 (C macro), 10
 HASH_SHA512 (C macro), 10
 hash_t (C++ type), 10
 hash_update (C++ function), 10

hash_update_async (C++ function), 10

K

key_id_t (C++ type), 10

M

module
 bio, 9
 crypto, 9
 plat, 12

P

plat
 module, 12
 plat_board_init (C++ function), 13
 plat_boot_reason (C++ function), 13
 plat_boot_reason_str (C++ function), 13
 plat_get_unique_id (C++ function), 13
 plat_get_us_tick (C++ function), 13
 plat_init (C++ function), 13
 plat_reset (C++ function), 13
 plat_wdog_kick (C++ function), 13