

Implementing Navigation Message Authentication on SBAS

SPENCER PAUL'S REU

Stanford University GPS Lab

6/1/21

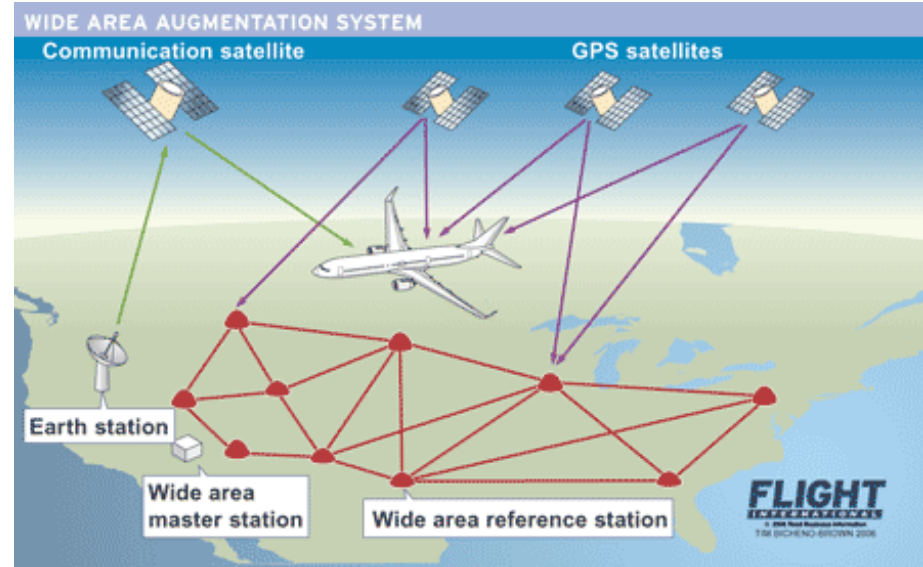
Background Information

SBAS, CRYPTOGRAPHY



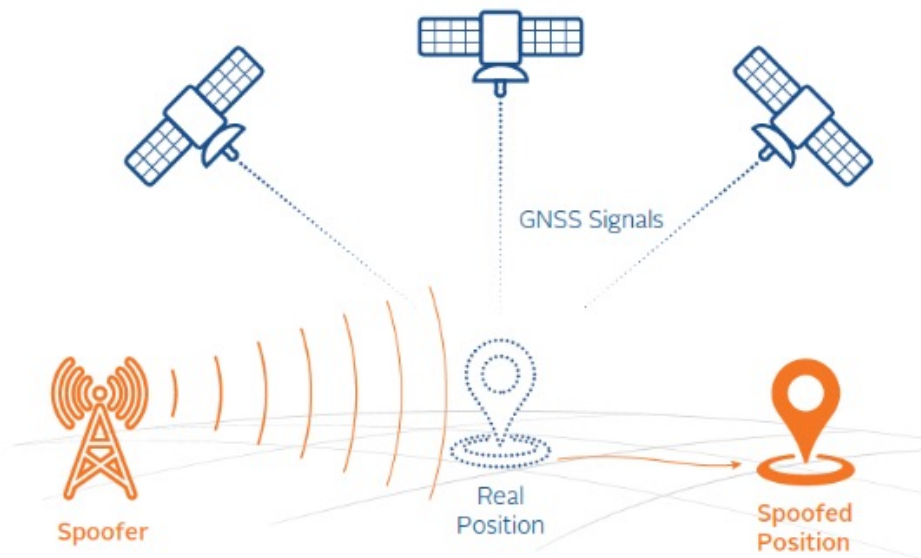
Satellite-based Augmentation System (SBAS)

- SBAS uses a collection of listening stations at **known** position to provide corrections
- Provides augmentation information for corrections of satellite position errors, clock/time errors and errors induced by delay of the signal while crossing the ionosphere.
- increases the accuracy with position errors below 1 meter



What is Spoofing?

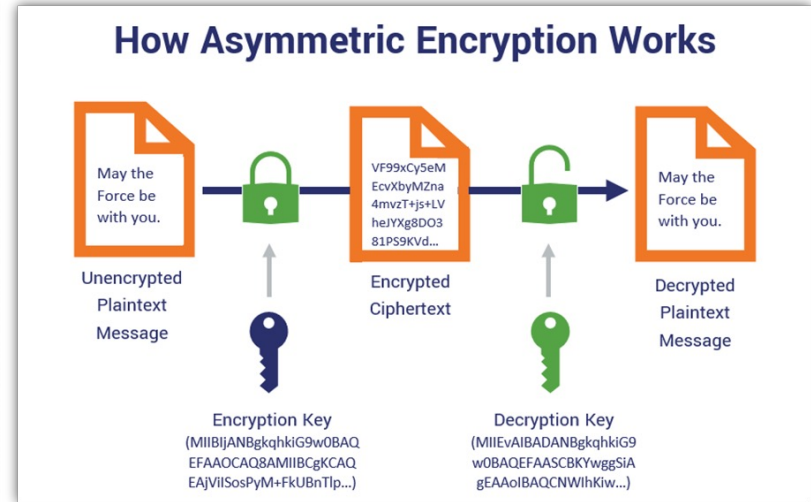
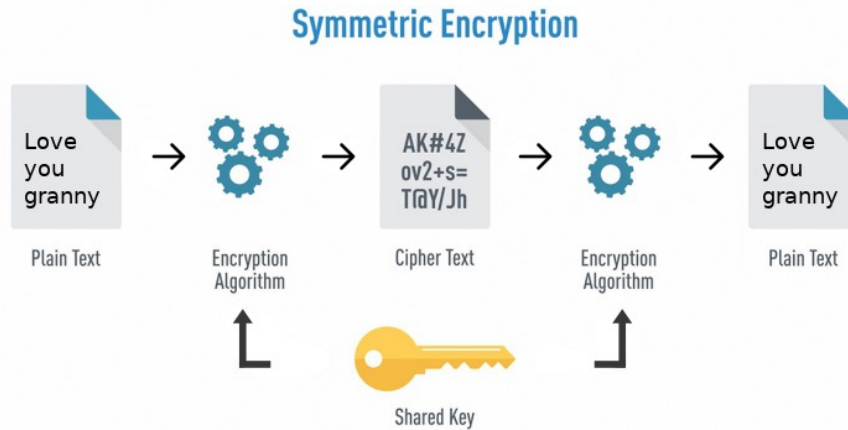
- malicious actors want to trick (spooft) GNSS users by manipulating their location
- Areas this could potentially occur -- smartphone to produce false Uber charges, mislead a passenger jet or autonomous vehicle



Authentication is used to prevent spoofing

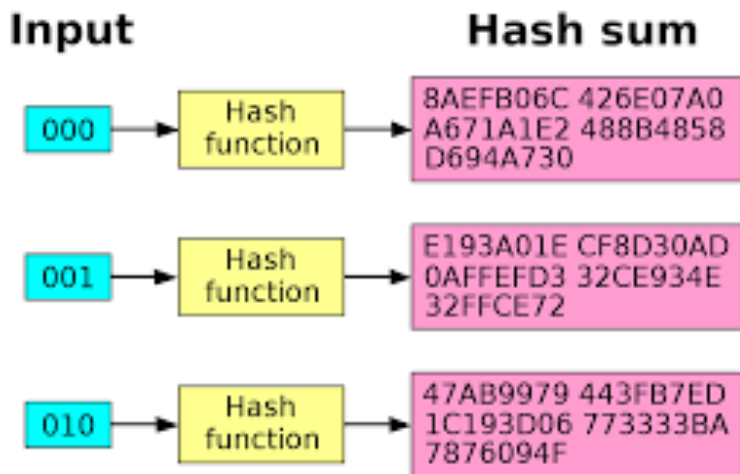
- Authentication is the process of determining if a claim is true
 - i.e gmail authenticates you are yourself when you log into your email
 - Stanford makes you scan an ID card to prove your identity when entering buildings
- This is distinct from encryption which encodes information that is not comprehensible without a specific key to decode it
- My work focused on implementing navigation message authentication (NMA) on SBAS to hinder spoofing

Asymmetric v Symmetric Encryption



Hash Functions

- Deterministic – each input has the same output every time
- One Way – it is very hard to determine input from hash
- Collision Free – no two inputs will have the same hash sum



Our Scheme

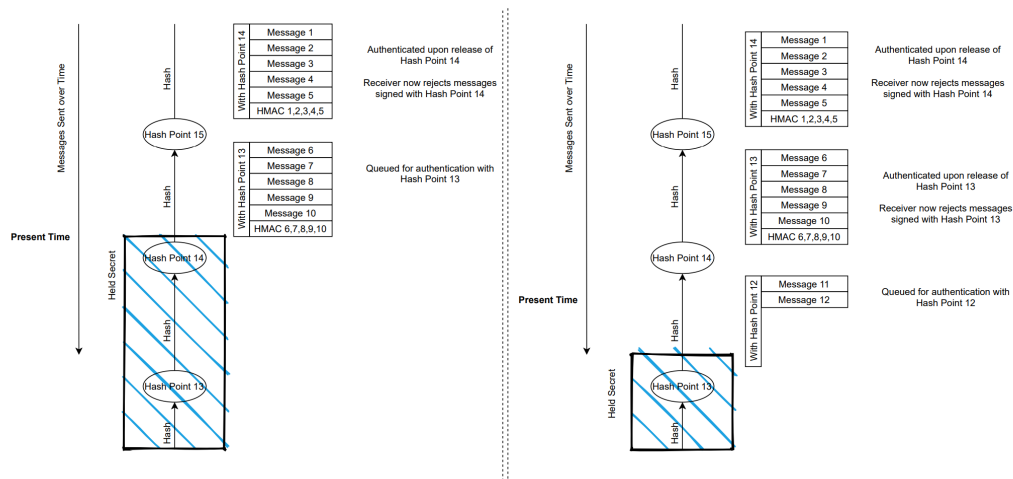
TESLA-ECDSA

Tesla-ECDSA Scheme

- TESLA must be used in tandem with ECDSA
 - TESLA authenticates the SBAS messages, and ECDSA authenticates SBAS's TESLA use as periodic maintenance.
 - We modify the SBAS message schedule by appending an extra message after every 5 messages that is used for authentication
- Requirements for design
 - [Jason help]

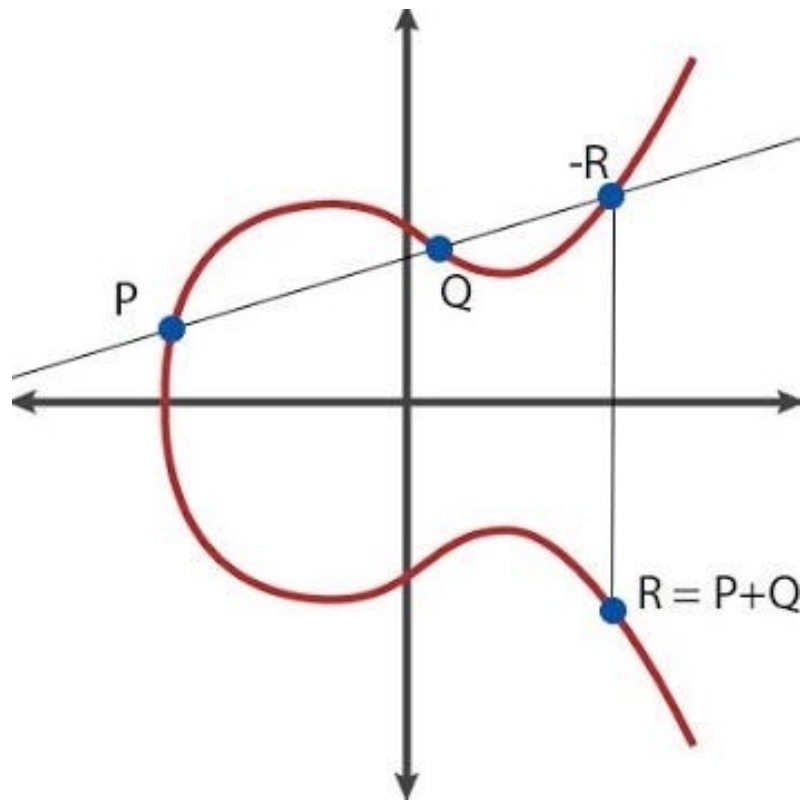
TESLA Portion of Scheme

- Very simple – only requires one cryptographic hash function ($h(\text{point})$)
- Works by generating a chain of hashes $p1 = h(p2)$, $p2 = h(p3)$, $p3 = h(p4)$
- $p4 \rightarrow p3 \rightarrow p2 \rightarrow p1$



ECDSA

- ECDSA stands for Elliptic Curve Digital Signing algorithm and is used to create a digital signature of data
- Used to verify the first hash point that is released
- This is also asymmetric and relies on the use of curves and mathematical equations to generate a public key that can verify a signature.



Zooming in on the actual messages

- MT50 – message containing authentication data
- HMAC – message authentication code for each message
 - $\text{key} = \text{xor}(\text{hp97}, \text{time})$
 - $\text{HMAC6} = \text{h}(\text{message}, \text{key})$
- 6-11 second delay before we can authenticate messages

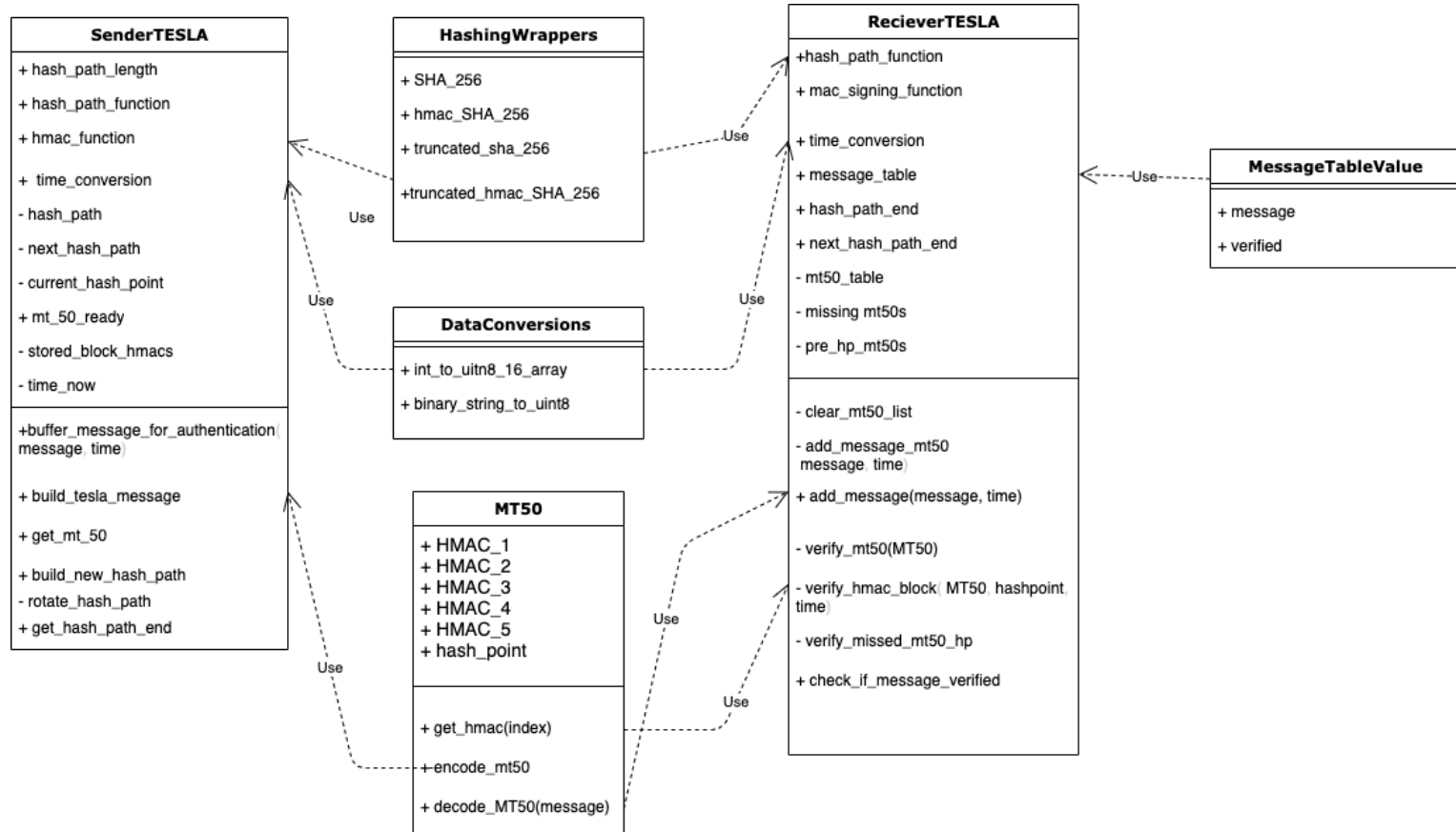
	SBAS Message 6					
	SBAS Message 7					
	SBAS Message 8					
	SBAS Message 9					
	SBAS Message 10					
MT50	HMAC6	HMAC7	HMAC8	HMAC9	HMAC10	Hash Point 98
	SBAS Message 11					
	SBAS Message 12					
	SBAS Message 13					
	SBAS Message 14					
	SBAS Message 15					
MT50	HMAC11	HMAC12	HMAC13	HMAC14	HMAC15	Hash Point 97
	SBAS Message 16					
	SBAS Message 17					
	SBAS Message 18					
	SBAS Message 19					
	SBAS Message 20					
MT50	HMAC16	HMAC17	HMAC18	HMAC19	HMAC20	Hash Point 96

Implementing the Scheme

Programing authentication into MAAST

- MAAST is a simulation on MATLAB of SBAS
- We implemented a full-stack SBAS simulation of our design into MAAST
- MAAST provides Monte-Carlo simulation results to evaluate how our design performs under message loss over the WAAS coverage area
- Our KPIs: time to first authenticated fix, time to authentication per message, and robustness to message loss

Class diagram of what I was coding this quarter



Edge cases in simulation added a lot of complexity

methods (Test)

```
function test_key_chain_construction(testCase) [] []

function test_M50_creation(testCase) [] []

function test_next_keychain(testCase) [] []

function test_sender_and_reciever(testCase) [] []

function test_manipulated_message(testCase) [] []

function test_manipulated_mt50(testCase) [] []

function test_missing_mt50(testCase) [] []

function test_swapped_mt50(testCase) [] []

function test_swapped_messages(testCase) [] []

function test_sender_and_reciever_keychain_rotation(testCase) [] []

function test_missing_message(testCase) [] []

function test_many_rotations(testCase) [] []

function test_long_string_missed_messages(testCase) [] []

function test_constructor_args(testCase) [] []

function test_1_arg_constructor(testCase) [] []
```

Adding missed message functionality

```
function verify_missed_mt50_hp(obj)
    largest_verified_key = max(cell2mat(keys(obj.mt50_table)));
    for i = 1:length(obj.missing_mt50)
        if isKey(obj.mt50_table, obj.missing_mt50(i) - 1) && ...
            largest_verified_key > obj.missing_mt50(i)

            count = largest_verified_key;
            new_hash = obj.mt50_table(largest_verified_key);
            new_hash = new_hash(1).hash_point;

            while count ~= obj.missing_mt50(i)
                new_hash = obj.key_chain_function(new_hash);
                count = count - 1;
            end

            if obj.missing_mt50(i) - 1 ~= 0
                mt50 = obj.mt50_table(obj.missing_mt50(i) - 1);
                m = mt50(1);
                t = mt50(2);

                obj.verify_hmac_block(m, new_hash, t);
            end
        end
    end
    obj.missing_mt50 = {};
end
```

Hash path rotation

```
function buffer_message_for_authentication(obj, message, time)
    message = DataConversions.binary_string_to_uint8(message);

    % keeping track of current hmac index and total messages
    obj.message_count = obj.message_count + 1;
    obj.hmac_num = obj.hmac_num + 1;

    % once we reach end of key chain - rotate the hash path
    if obj.key_chain_length - obj.current_block <= 0
        % store last element of key chain for next mt50
        obj.stored_block{6} = obj.key_chain(1);
        obj.rotate_hash_path();
        obj.current_block = 1;
    end
```


Coding Practices



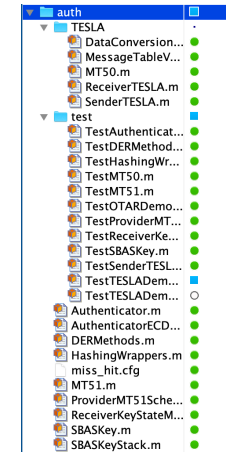
Unit Testing

- Unit Testing was critical to prevent nasty bugs in the future when we integrated code into MAAST
- Every line of code had to be unit tested to pass our build check
- We had ~1500 lines of code dedicated to unit testing ~1500 lines of actual code

matlab unittest coverage report

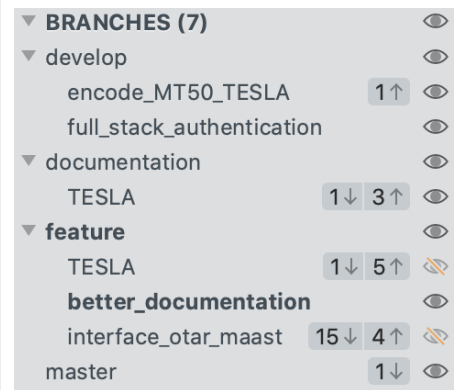
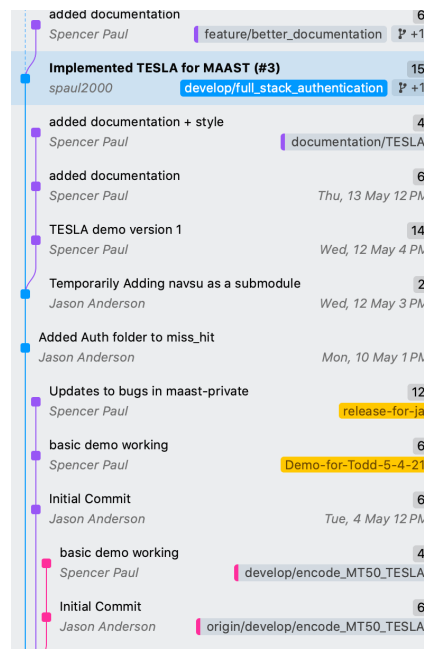
File	Coverage	Lines	Branches		Missing
All files	1%	1%	NaN%	✗	
Authenticator	100%	100%	NaN%	✓	
AuthenticatorECDSA	100%	100%	NaN%	✓	
HashingWrappers	100%	100%	NaN%	✓	
assert_matlab_checkcode	0%	0%	NaN%	✗	4-30
execute_matlab_tests_into_xml	0%	0%	NaN%	✗	13-32
TestAuthenticatorECDSA	100%	100%	NaN%	✓	
TestHashingWrappers	100%	100%	NaN%	✓	

Minimum allowed coverage is 100%



Using GitHub for organized version control

- Helped Jason and myself collaborate effectively throughout the quarter
- Allowed us to keep a linear version history
- Prevents merge conflicts and allowed us to see exactly where our code diverged



Other Coding Checks

- Mandatory code review for PR approval
- Style Checks using Miss Hit

Review requested
Review has been requested on this pull request. It is not required to merge. [Learn more.](#) [Show all reviews](#)

2 pending reviewers

All checks have passed
4 successful checks [Hide all checks](#)

- ✓ **Matlab Code Build Checker / MISS_HIT Linter (push)** Successful in 12s [Details](#)
- ✓ **Matlab Code Build Checker / Matlab Checkcode Linter (push)** Successful in 13s [Details](#)
- ✓ **Matlab Code Build Checker / Matlab Runtests (push)** Successful in 5m [Details](#)
- ✓ **Matlab Code Build Checker / Matlab Runtests Results (push)** — 75 tests run, 0 skipped, 0 failed. [Details](#)

This branch has conflicts that must be resolved
[Use the command line](#) to resolve conflicts before continuing. [Resolve conflicts](#)

Conflicting files
README.md

Squash and merge You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

jand271 requested changes on May 3 [View changes](#)

auth/TESLA/ConversionWrappers.m Outdated [Show resolved](#)

auth/TESLA/ConversionWrappers.m Outdated [Hide resolved](#)

```
2 +  
3 + methods (Static)  
4 +  
5 + function uint8_array = time_to_uint8(time)
```

jand271 on May 3
"int_to_uint8_array"...remove notation of time. Also add size, or name function
"int_to_uint8_16_array"

[Reply...](#)

Unresolve conversation spaul2000 marked this conversation as resolved.

auth/TESLA/ConversionWrappers.m Outdated [Show resolved](#)

auth/TESLA/MT50.m Outdated [Show resolved](#)