Crypto.com Chain Threat Modeling

	Risk	Severity	Exploitability	Note	Mitigation
Spoofing					
1.1 Impersonate a council node					
1.1.1 Get the block-signing secret key					
1.1.1.1 Decrypt the encrypted block-signing secret key	3.2	5	2		Restrict physical and remote access to the server. Tendermint core itself is not running inside enclave. But the signing function should be executed inside a enclave or HSM. A remote HSM will introduce extra network latency and therefore a local enclave is recommended. However, even with enclave or HSM, Tendermint still needs a password to connect to the enclave or HSM, and the password should not be stored in any config file and should be manually typed during system startup
1.1.1.2 Scan the key from the memory directly	3.9	5	3	See RAMBleed	Either use HSM, USB Key or clear memory immediately after use
1.1.1.3 Read the block-signing secret key using side channel attacks	3.5	5	2.5	See power-analysis, timing attacks, etc.	Make sure the encryption algorithm is resistant to side channel attacks
1.1.2 Use the block-signing secret key					
1.1.2.1 Login to enclave or HSM to use the key directly	3.9	5	3	Also see 1.1.1.1	The HSM/USB key/enclave password should be manually typed during startup. Enclave should be local access only, USB Key should be access controlled, HSM should be IP restricted
1.1.3 Guess the block-signing secret key	0.7	5	0.1		Make sure secure random is used when generating the secret key
1.1.4 Forge a valid signature without the key	0.7	5	0.1		
1.1.5 Mallet a valid signature into another purpose	0.7	5	0.1		
1.2 Impersonate a non-council node					
1.2.1 Get the P2P communction secret key					
1.2.1.1 Decrypt the encrypted P2P communction secret key	1.7	1.5	2	Restrict physical and remote access to the server	Non-council node / comm key probably won't use HSM/USB Key. But we can make sure the password is (long) strong enough
1.2.1.2 Scan the key from the memory directly	2.1	1.5	3	Restrict physical and remote access to the server	Clear memory immediately after use
1.2.1.3 Read the P2P communction secret key using side channel attacks	1.9	1.5	2.5	Restrict physical and remote access to the server	Make sure the encryption algorithm is resistant to side channel attacks
1.2.2 Guess the secret key	0.4	1.5	0.1		Make sure secure random is used when generating the secret key
1.2.3 Forge a valid signature without the key	0.4	1.5	0.1		
1.2.4 Mallet a valid signature into another purpose	0.4	1.5	0.1		
1.3 Impersonate a wallet end-user					
1.3.1 Spend Authorization Secret Key					
1.3.1.1 Get the spend authorization secret key					
1.3.1.1.1 Decrypt the encrypted spend authorization secret key	2.4	3	2		The wallet app should support hardware wallet or using strong password to encrypt the private key. The app should ask for the password when it needs to sign a transaction.
1.3.1.1.2 Scan the key from the memory directly	3.0	3	3		Clear memory immediately after use
1.3.1.1.3 Read the spend authorization secret key using side channel attacks	2.7	3	2.5		Make sure the encryption algorithm is resistant to side channel attacks
1.3.1.2 Guess the spend authorization secret key	0.5	3	0.1		Make sure secure random is used when generating the secret key
1.3.1.3 Forge a valid signature without the key	0.5	3	0.1		
1.3.1.4 Mallet a valid signature into another purpose	0.5	3	0.1		
1.3.2 View Key					
1.3.2.1 Get the view key					
1.3.2.1.1 Decrypt the encrypted view key	2.0	2	2		The wallet app will need to store the private key in a config file and encrypted with a strong password. The app should ask for the password during startup.
1.3.2.1.2 Scan the key from the memory directly	2.4	2	3		No way to clear the memory "after" use because it will need to keep using it after startup.
1.3.2.1.3 Read the view key using side channel attacks	2.4	2	3		Make sure the encryption algorithm is resistant to side channel attacks
1.3.2.3 Guess the view key	0.4	2	0.1		Make sure secure random is used when generating the secret key
1.3.2.4 Forge a valid signature without the key	0.4	2	0.1		
1.3.2.5 Mallet a valid signature into another purpose	2.4	2	3		The wallet will need to sign a message to prove to the full node about his identity. Security safeguard should be in place to prevent replay attack.
1.4 Impersonate a valid enclave					
1.4.1 Exploit 0day SGX bug	2.4	4	1.5		
1.4.2 Impersonate Intel IAS					
1.4.2.1 Redirect IAS traffic to another host	0.6	4	0.1	Protected by IAS cert	
Tampering					
2.1 Modify system code directly					
2.1.1 Modify ABCI runtime code	0.5	3	0.1	Protected by code signing and enclave	
2.1.2 Modify ABCI source code					
2.1.2.1 Modify ABCI Github source code	3.9	5	3		Use strong 2FA for Github account, use protected branches.
2.1.2.2 Modify ABCI code dependency	3.9	5	3		Fix dependency version, minimize dependent libraries, avoid libraries with many transitive dependencies and thoroughly review the included code
2.1.2.3 Use the code signing key to sign an unauthorized code	3.9	5	3		Need to have a secure code signing facility, recommended using strong hardware token. Also need to think about Council node signatures.
2.1.3 Modify Tendermint core code	1.7	1	3		Council node require staking, non-council node has no impact on the consensus protocol
2.2 Modify blockchain header	0.7	5	0.1	Protected by validator signatures	
2.3 Modify confirmed transaction details					
2.3.1 Modify levelDB directly	2.6	2	3.5	For UTXO, the system need to maintain if an UTXO has been spent. Unless the system rescan the whole blockchain everytime, there must be some kind of cache. And this data should be part of levelDB. A malicious hacker can modify the data by marking an UTXO as NOT spent, but it will only affect his own data.	Except for the UTXO flag, the system should re-check the hash/signature and other data whenever possible
2.4 Modify pending transaction details	0.6	4	0.1	Protected by wallet signature	

	Risk	Severity	Exploitability	Note	Mitigation
2.5 Modify communication between nodes					
2.5.1 Modify traffic data	0.6	4	0.1	Protected by Station-to-Station protocol, and even if a hacker can modify the traffic, messages are protected by signatures	
2.5.1 Drop consensus messages between nodes	2.0	1	4	There should be multiple paths between any two validators. And even a pre-vote or pre-commit message is dropped, the timeout mechanism will come to play and the system will continue to work. The system is more stable when there are more non-validator nodes	Incentivize more people to setup their own full node
2.5.2 Drop broadcasted transaction messages	2.0	1	4		The wallet app should broadcast it's transaction to multiple nodes
2.6 Modify communication between Tendermint and ABCI					
2.6.1 Connect directly to the ABCI	1.0	1	1	Injecting data directly into ABCI can interfere the state data. Data still need to be signed but data does not need to have consensus.	Make sure the ABCI can only be connected by the local Tendermint core process. This can be done by setting the ACL of the unix domain socket
Repudiation					
3.1 Council node repudiate a committed block	0.7	5	0.1	Protected by signatures	
3.2 Wallet end-user repudiate a transaction	0.7	5	0.1	Protected by signatures	
3.3 Recipient refuse to confirm a payment	0.5	3	0.1	Sender can publish the transaction in plain text, a 3rd party verifier can verify the transaction id was included in a certain block	Unless the payer includes her own viewkey in the transaction, there is no way to retrieve the plain text transaction. Therefore, the payer should backup all her transactions for dispute management.
Information Disclosure					
4.1 Disclose transaction details					
4.1.1 Read the secret key using side channel attacks	3.2	4	2.5		Make sure the encryption algorithm is resistant to side channel attacks. And since we will use AES, using CPU built-in AES op-code might be the best solution.
4.1.2 Read the secret key by abusing a SGX bug	2.8	4	2		Not much we can do to "prevent" a SGX bug. But in order to contain the damage, we need to (1) be able to change to the key, (2) be able to refuse sending a new key to an "old" node
4.1.3 Read plain text transaction due to a bug in Station-to-Station protocol	1.6	2.5	1	End user wallet broadcast the transaction in plain text	
4.1.4 Read plain text transaction by modifying the Tendermint core runtime	3.5	3	4	End user wallet broadcast the transaction in plain text	Two solutions: (1) sender need to use PKI to encrypt the message to make sure only the trusted enclave, not the Tendermint core, can read the data. The secret value should be bound to the attestation report. (2) wallet send plain text transaction to enclave (bypassing Tendermint) directly. But in this case, the enclave would than have public exposure.
4.2 Disclose wallet end-user location					
4.2.1 Get wallet IP by modifying the Tendermint core runtime	1.7	1	3	This will be much harder if the submitted transaction is encrypted (see 4.1.4)	(1) Only submit transaction to your own full node, (2) submit transactions to different full nodes (assuming there are many full nodes)
4.2.1 Get wallet IP by traffic monitoring	0.3	1	0.1	Without tx plain text, traffic monitoring can at most associate txid to IP, not wallet addr	
Denial of Service					
5.1 DoS a council node					
5.1.1 Directly DoS the council node				One needs to know the IP before he can launch the attack, see 4.2	
5.1.2 DoS the sentries of the council node	3.0	3.5	2.5	One may be able to deduce the sentry nodes by monitoring Tendermint traffic	1. The more sentries the better 2. Be able to spawn new sentries when needed
5.2 DoS a non-council node	1.6	1	2.5	Maybe a wallet (an acquirer) is connected to a dedicated full node	
5.3 DoS Intel IAS	1.4	4	0.5	The system is still vulnerable to DoS even if we upgrade to use DCAP. However, when the system is under DoS, existing nodes will still be able to operate, and usually, DoS will not last too long, the risk is still acceptable.	