# Incentives in Casper the Friendly Finality Gadget

**Vitalik Buterin**
Ethereum Foundation

August 18, 2017

## Abstract

We give an introduction to the incentives in the Casper the Friendly Finality Gadget protocol, and show how the protocol behaves under individual choice analysis, collective choice analysis and griefing factor analysis. We define a "protocol utility function" that represents the protocol's view of how well it is being executed, and show the connection between the incentive structure that we present and the utility function. We show that (i) the protocol is a Nash equilibrium assuming any individual validator's deposit makes up less than $\frac{1}{3}$ of the total, (ii) in a collective choice model, where all validators are controlled by one actor, harming protocol utility hurts the cartel's revenue, and there is an upper bound on the ratio between the reduction in protocol utility from an attack and the cost to the attacker, and (iii) the griefing factor can be bounded above by 1, though we will prefer an alternative model that bounds the griefing factor at 2 in exchange for other benefits.

## 1 Introduction

[define blocks, epochs]

In the Casper protocol, there is a set of validators, and in each epoch validators have the ability to send two kinds of messages:

| **Notation** | **Description** |
|---|---|
| h | the hash to justify |
| $e$ | the current epoch |
| $h_\star$ | the most recent justified hash |
| $e_\star$ | the epoch containing hash $h_\star$ |
| $\mathcal{S}$ | signature from the validator's private key |

(a) PREPARE format

| **Notation** | **Description** |
|---|---|
| h | the hash to finalize |
| $e$ | the current epoch |
| $\mathcal{S}$ | signature from the validator's private key |

(b) COMMIT format

Table 1: The schematic of the PREPARE and COMMIT messages. [it's unclear to me why we need $e_\star$.]

Each validator has a *deposit size*; when a validator joins their deposit size is equal to the number of coins that they deposited, and from there on each validator's deposit size rises and falls with rewards and penalties.

For the rest of this paper, when we say "$^2/_3$ of validators", we are referring to a *deposit-weighted* fraction; that is, a set of validators whose sum deposit size equals to at least $\frac{2}{3}$ of the total deposit size of the entire set of validators. We also use "$^2/_3$ Prepares" and "$^2/_3$ Commits" as shorthand for "$\frac{2}{3}$ of deposit-weighted validators sent Prepares/Commits".

Every hash h has one of three possible states: *fresh*, *justified*, and *finalized*. Every hash starts as *fresh*. The hash at the beginning of the current epoch converts from fresh to *justified* if, during the current epoch $e$, $^2/_3$ Prepares are sent of the form

$$[\textbf{PREPARE}, e, \mathrm{h}, e_\star, \mathrm{h}_\star, \mathcal{S}] \tag{1}$$

for some specific $e_\star$ and $\mathrm{h}_\star$. A hash converts from justified to *finalized*, if $^2/_3$ Commits

$$[\textbf{COMMIT}, e, \mathrm{h}, \mathcal{S}] \ , \tag{2}$$

for the same $e$ and h as in (4). The h is the block hash of the block at the start of the epoch. A hash h being justified entails that all fresh (non-finalized) ancestor blocks are also justified. A hash h being finalized entails that all ancestor blocks are also finalized, regardless of whether they were previously fresh or justified. An "ideal execution" of the protocol is one where, at the start of every epoch, every validator Prepares and Commits the first blockhash of each epoch, specifying the same $e_\star$ and $\mathrm{h}_\star$. We wish to incentivize this ideal execution.

Possible deviations from this ideal execution that we want to minimize or avoid include:

- Violating any of the four slashing conditions. [1]
  1. **PREPARE_REQ**. If a validator Prepares specifying a the $\mathrm{h}_\star$ is not *justified*, the validator is penalized.
  2. **NO_DBL_PREPARE**. If a validator publishes two nonidentical Prepares with the same $e$ value, the validator is penalized. This is equivalent to that each validator may Prepare to exactly one (h, $e_\star$, $\mathrm{h}_\star$) triplet per epoch.
  3. **COMMIT_REQ**. If a validator Commits an unjustified hash, the validator's is penalized.
  4. **PREPARE_COMMIT_CONSISTENCY**. Given a Prepare-Commit pair of a single hash, from a single validator of the form,

     $$[\textbf{PREPARE}, e_1, \mathrm{h}, e_\star, \mathrm{h}_\star, \mathcal{S}]$$

     $$[\textbf{COMMIT}, e_2, \mathrm{h}, \mathcal{S}] \ ,$$

     the epochs must satisfy $e_\star < e_1 < e_2$. Otherwise, the validator is penalized.
- By the end of epoch $e$, the first blockhash of epoch $e$ is not yet finalized.

Each validator only see the blockchain's own history, including messages that were passed in. [Are Commits/Prepares stored on-chain?]

The blockchain state stores the latest justified hash, $\mathrm{h}_{\mathrm{LJ}}$, and only reward Prepares whose $e_\star$ and $\mathrm{h}_\star = \mathrm{h}_{\mathrm{LJ}}$. These two techniques will help to coordinate validators toward Preparing and Committing a single hash h with a single source $\mathrm{h}_\star$.

Let TD be the current *total amount of deposited coins*, and $e - e_{\mathrm{LF}}$ be the number of epochs since the last finalized epoch.

## 2 Rewards and Penalties

We define the following nonnegative functions,

- $\mathrm{BIR}(\mathrm{TD})$: returns the base interest rate paid to a validator, taking as an input the current total quantity of deposited coins.
- $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}})$: returns the "base penalty constant"—a value expressed as a percentage rate that is used as the scaling factor for all penalties; for example, if at the current time $\mathrm{BP}(\cdot, \cdot, \cdot) = 0.001$, then a penalty of 1.5 means a validator loses 0.15% of their deposit. Takes as inputs the current total quantity of deposited coins TD, the current epoch $e$ and the last finalized epoch $e_{\mathrm{LF}}$. Note that in a perfect protocol execution, $e - e_{\mathrm{LF}} = 1$.
- $\mathrm{NPCP}(\alpha)$ ("non-prepare collective penalty"): if $\alpha$ of validators ($0 \le \alpha \le 1$) are not seen to have Prepared during an epoch, then *all* validators are charged a penalty of $\mathrm{NCCP}(\alpha)$. NPCP must be monotonically increasing, and satisfy $\mathrm{NPCP}(0) = 0$.
- $\mathrm{NCCP}(\alpha)$ ("non-commit collective penalty"): if $\alpha$ of validators ($0 \le \alpha \le 1$) are not seen to have Committed during an epoch, and that epoch had a justified hash so any validator *could* have Committed, then all validators are charged a penalty proportional to $\mathrm{NCCP}(\alpha)$. NCCP must be monotonically increasing, and satisfy $\mathrm{NCCP}(0) = 0$.

We also define the following nonnegative constants:

- NPP ("non-prepare penalty"): the penalty for not preparing. $\mathrm{NPP} > 0$.
- NCP ("non-commit penalty"): the penalty for not committing, if there was a justified hash which the validator *could* have Committed. NCP is a constant and $\mathrm{NCP} > 0$.

All functions return a nonnegative scalar with no units. Technically these values can exceed 1.0, but in practice they will be rarely exceed 0.01.

Note that a validator Preparing/Committing it won't incur a NPP/NCP; it could be the case that either because of very high network latency or a malicious majority censorship attack, the Prepares and Commits are not included into the blockchain in time and so the incentivization mechanism does not see them. For NPCP and NCCP similarly, the $\alpha$ input is the proportion of validators whose Prepares and Commits are *not visible*, not the proportion of validators who *tried to send* Prepares/Commits.

When we talk about preparing and committing the "correct value", we are referring to the hash $\mathrm{h}$ and the parent epoch $e_\star$ and parent hash $\mathrm{h}_\star$.

We now define the following reward and penalty schedule, which runs at the *end* of each epoch.

1. All validators get a reward of $\mathrm{BIR}(\mathrm{TD})$ (e.g., if $\mathrm{BIR}(\mathrm{TD}) = 0.0002$ then a validator with $10,000$ coins deposited gets a per-epoch reward of 2 coins)
2. If the protocol does not see a Prepare from a given validator during the epoch, the validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NPP}$
3. If the protocol does not see a Commit from a given validator during the epoch, and a Prepare was justified so a Commit *could have* been seen, they are penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NCP}$.
4. If the protocol saw Prepares from proportion $p$ validators during the epoch, *every* validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NPCP}(1 - p)$
5. If the protocol saw Commits from proportion $p$ validators during the epoch, and a Prepare was justified so a validator *could have* Commited, then *every* validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NCCP}(1 - p)$.
6. The blockchain's recorded $e_{\mathrm{LF}}$ and $\mathrm{h}_{\mathrm{LF}}$ are updated to the latest values. [correct?]

This is the entirety of the incentivization structure. We will define the functions and constants later, attempting to derive the specific values from first principles and desired objectives.

## 3 Three theorems

We seek to prove the following:

**Theorem 1** ([First theorem]). *If no validator has more than $\frac{1}{3}$ of the total deposit, i.e.,* $\max_i(D) \leq \frac{\mathrm{TD}}{3}$*, then Preparing the last blockhash of each epoch and Committing the* $\mathrm{h}_{\mathrm{LJ}}$ *is a Nash equilibrium. (Section 3.1)*

**Theorem 2** ([Second theorem]). *Even if all validators collude, the ratio between the harm inflicted and the penalties paid by validators is bounded above by some constant. Note that this requires a measure of "harm inflicted"; we discuss this Section 3.2.*

**Theorem 3** ([Third theorem]). *The* griefing factor*, the ratio of the penalty incurred by the victims of an attack and penalty incurred by the attackers. Even when the attackers hold a majority of the total deposit, the griefing factor is at most 2. (Section 3.3)*

### 3.1 Individual choice analysis

The individual choice analysis is simple. Suppose that during epoch $e$ the proposal mechanism Prepares a hash h and the Casper incentivization mechanism specifies some $e_\star$ and $\mathrm{h}_\star$. Because, as per definition of the Nash equilibrium, we are assuming that all validators except for the validator that we are analyzing are following the equilibrium strategy, we know that $\geq \frac{2}{3}$ of validators Prepared in the last epoch and so $e_\star = e - 1$, and $\mathrm{h}_\star$ is the direct parent of h.

Hence, the PREPARE_COMMIT_CONSISTENCY slashing condition poses no barrier to preparing $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$. Since, in epoch $e$, we are assuming that all other validators *will* Prepare these values and then Commit h, we know h will be a hash in the main chain, and so a validator will pay a penalty if they do not Prepare $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$, and they can avoid the penalty if they do Prepare these values.

We are assuming there are $\frac{2}{3}$ Prepares for $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$, and so PREPARE_REQ also poses no barrier to committing h. Committing h allows a validator to avoid NCP. Hence, there is an economic incentive to Commit h. This shows that, if the proposal mechanism succeeds at presenting to validators a single primary choice, Preparing and Committing the value selected by the proposal mechanism is a Nash equilibrium.

[Put two 2x2 tables here for the trade-offs in Preparing and Committing h?]

### 3.2 Collective choice model

To model the protocol in a collective-choice context, we first define a *protocol utility function*. The protocol utility function quantifies "how well the protocol execution is doing". Although our specific protocol utility function cannot be derived from first principles, we can intuitively justify it.

We define our protocol utility function as,

$$U = \sum_{k=0}^{e} -\log_2\left[k - e_{\mathrm{LF}}\right] - MF \,. \tag{3}$$

Where:

- $e$ is the current epoch, starting from 0.
- $e_{\mathrm{LF}}$ is the index of the last finalized epoch. [To be clear, does the $e_{\mathrm{LF}}$ change with the term $k$, or is it fixed?]
- $M$ is a very large constant.
- $F$ is an Indicator function. It returns 1 if a safety failure has taken place, otherwise 0. [It'd be nice to get a descripton of the conditions that lead to a "safety failure".]

The second term in the function is easy to justify: safety failures are very bad. The first term is trickier. To see how the first term works, consider the case where every epoch such that $e$ mod $N$, for some $N$, is zero is finalized and other epochs are not. The average total over each $N$-epoch slice will be roughly $\sum_{i=1}^{N} -\log_2(i) \approx N * \left[\log_2(N) - \frac{1}{\ln(2)}\right]$. Hence, the utility per block will be roughly $-\log_2(N)$. This basically states that a blockchain with some finality time $N$ has utility roughly $-\log(N)$, or in other words *increasing the finality time of a blockchain by a constant factor causes a constant loss of utility.* The utility difference between 1 minute and 2 minute finality is the same as the utility difference between 1 hour and 2 hour finality.

This can be justified in two ways. First, one can intuitively argue that a user's psychological discomfort of waiting for finality roughly matches a logarithmic schedule. At the very least, the difference between 3600 sec and 3610 sec finality feels much more negligible than the difference between 1 sec and 11 sec finality, and so the claim that the difference between 10 sec and 20 sec finality is similar to the difference between 1 hour finality and 2 hour finality does not seem farfetched. [1]

Now, we need to show that, for any given total deposit size, $\frac{loss\_to\_protocol\_utility}{validator\_penalties}$ is bounded. There are two ways to reduce protocol utility: (i) cause a safety failure, and (ii) prevent finality by having $> \frac{1}{3}$ [$\geq$ *or* $>$?] of validators not Prepare or Commit to the same hash. Causing a safety failure violates one of the slashing conditions and thus ensures a large loss in deposits. In the second case, in a chain that has not been finalized for $e - e_{\text{LF}}$ epochs, the penalty to attackers is at least,

$$\min\left[\text{NPP} * \frac{1}{3} + \text{NPCP}\left(\frac{1}{3}\right), \text{NCP} * \frac{1}{3} + \text{NCCP}\left(\frac{1}{3}\right)\right] * \text{BP}(\text{TD}, e - e_{\text{LF}}) \,. \tag{4}$$

To enforce a ratio between validator losses and loss to protocol utility, we set,

$$\text{BP}(\text{TD}, e - e_{\text{LF}}) \equiv \frac{k_1}{\text{TD}^p} + k_2 * \lfloor \log_2(e - e_{\text{LF}}) \rfloor \,. \tag{5}$$

what is $p$ in the in the above equation?

The first term serves to take profits for non-committers away; the second term creates a penalty which is proportional to the loss in protocol utility.

This connection between validator losses and loss to protocol utility has several consequences. First, it establishes that harming the protocolexecution is costly, and harming the protocol execution more costs more. Second, it establishes that the protocol approximates the properties of a potential game [cite]. Potential games have the property that Nash equilibria of the game correspond to local maxima of the potential function (in this case, protocol utility), and so correctly following the protocol is a Nash equilibrium even in cases where a coalition has more than $\frac{1}{3}$ of the total validators. Here, the protocol utility function is not a perfect potential function, as it does not always take into account changes in the *quantity* of prepares and commits whereas validator rewards do, but it does come close.

### 3.3 Griefing factor analysis

Griefing factor analysis is important because it provides a way to quanitfy the risk to honest validators. In general, if all validators are honest, and if network latency stays below half [half, right?] the length of an epoch, then validators face zero penalties to their respective deposits. In the case where malicious validators exist, however, they can interfere in the protocol in ways that penalize themselves as well as honest validators.

We define the "griefing factor" as,

$$\mathcal{GF}(\eth, C) \equiv \max_{S \in strategies(T \setminus C)} \frac{loss(C)}{\min[0, loss(Players \setminus C)]} \,. \tag{6}$$

I need to work on this equation more. I don't like it yet.

**Definition 1.** *A strategy used by a coalition in a given mechanism has a* griefing factor $B$ *if it can be shown that this strategy imposes a loss of* $B * x$ *to those outside the coalition at the cost of a loss of* $x$ *to those*

[1] One can look at various blockchain use cases, and see that they are roughly logarithmically uniformly distributed along the range of finality times between around 200 miliseconds ("Starcraft on the blockchain") and one week (land registries and the like). [add a citation for this or delete.]

*inside the coalition. If all strategies that cause deviations from some given baseline state have griefing factors less than or equal to some bound $B$, then we call $B$ a* griefing factor bound*.* *[I plan to write this in terms of classical game theory.]*

A strategy that imposes a loss to outsiders either at no cost to a coalition, or to the benefit of a coalition, is said to have a griefing factor of infinity. Proof of work blockchains have a griefing factor bound of infinity because a 51% coalition can double its revenue by refusing to include blocks from other participants and waiting for difficulty adjustment to reduce the difficulty. With selfish mining, the griefing factor may be infinity for coalitions of size as low as 23.21%. [**?**]

Figure 1: Plotting the griefing factor as a function of the proportion of players coordinating to grief.

Let us start off our griefing analysis by not taking into account validator churn, so the validator set is always the same. In Casper, we can identify the following deviating strategies:

1. A minority of validators do not prepare, or Prepare incorrect values.
2. (Mirror image of 1) A censorship attack where a majority of validators does not accept prepares from a minority of validators (or other isomorphic attacks such as waiting for the minority to Prepare hash $H_1$ and then preparing $H_2$, making $H_2$ the dominant chain and denying the victims their rewards).
3. A minority of validators do not commit.
4. (Mirror image of 3) A censorship attack where a majority of validators does not accept commits from a minority of validators.

Notice that, from the point of view of griefing factor analysis, it is immaterial whether or not any hash in a given epoch was justified or finalized. The Casper mechanism only pays attention to finalization in order to calculate $\text{BP}(D, e, e_{\text{LF}})$, the penalty scaling factor. This value scales penalties evenly for all participants, so it does not affect griefing factors.

Let us now analyze the attack types:

In general, we see a perfect symmetry between the non-Commit case and the non-Prepare case, so we can assume $\frac{\text{NCCP}(\alpha)}{\text{NCP}} = \frac{\text{NPCP}(\alpha)}{\text{NPP}}$. Also, from a protocol utility standpoint, we can make the observation that seeing $\frac{1}{3} \le p_c < {}^2\!/_3$ commits is better than seeing fewer commits, as it gives at least some economic security against finality reversions, so we want to reward this scenario more than the scenario where we get $\frac{1}{3} \le p_c < {}^2\!/_3$ prepares. Another way to view the situation is to observe that $\frac{1}{3}$ non-prepares causes *everyone* to non-commit, so it should be treated with equal severity.

In the normal case, anything less than $\frac{1}{3}$ commits provides no economic security, so we can treat $p_c < \frac{1}{3}$ commits as equivalent to no commits; this thus suggests $\text{NPP} = 2 * \text{NCP}$. We can also normalize $\text{NCP} = 1$.

| **Attack** | **Amount lost by attacker** | **Amount lost by victims** |
|---|---|---|
| Minority of size $\alpha < \frac{1}{2}$ non-prepares | $\text{NPP} * \alpha + \text{NPCP}(\alpha) * \alpha$ | $\text{NPCP}(\alpha) * (1 - \alpha)$ |
| Majority censors $\alpha < \frac{1}{2}$ prepares | $\text{NPCP}(\alpha) * (1 - \alpha)$ | $\text{NPP} * \alpha + \text{NPCP}(\alpha) * \alpha$ |
| Minority of size $\alpha < \frac{1}{2}$ non-commits | $\text{NCP} * \alpha + \text{NCCP}(\alpha) * \alpha$ | $\text{NCCP}(\alpha) * (1 - \alpha)$ |
| Majority censors $\alpha < \frac{1}{2}$ commits | $\text{NCCP}(\alpha) * (1 - \alpha)$ | $\text{NCP} * \alpha + \text{NCCP}(\alpha) * \alpha$ |

Now, let us analyze the griefing factors, to try to determine an optimal shape for NCCP. The griefing factor for non-committing is,

$$\mathcal{GF} = \frac{(1-\alpha) * \text{NCCP}(\alpha)}{\alpha * (1 + \text{NCCP}(\alpha))} \,. \tag{7}$$

The griefing factor for censoring is the inverse of this. If we want the griefing factor for non-committing to equal one, then we could compute:

$$\begin{aligned}
\alpha * (1 + \text{NCCP}(\alpha)) &= (1-\alpha) * \text{NCCP}(\alpha) && (8)\\
\frac{1 + \text{NCCP}(\alpha)}{\text{NCCP}(\alpha)} &= \frac{1-\alpha}{\alpha} && (9)\\
\frac{1}{\text{NCCP}(\alpha)} &= \frac{1-\alpha}{\alpha} - 1 && (10)\\
\text{NCCP}(\alpha) &= \frac{\alpha}{1-2\alpha} && (11)
\end{aligned}$$

Note that for $\alpha = \frac{1}{2}$, this would set the NCCP to infinity. Hence, with this design a griefing factor of 1 is infeasible. We *can* achieve that effect in a different way - by making NCP itself a function of $\alpha$; in this case, $\text{NCCP} = 1$ and $\text{NCP} = \max[0, 1 - 2 * \alpha]$ would achieve the desired effect. If we want to keep the formula for NCP constant, and the formula for NCCP reasonably simple and bounded, then one alternative is to set $\text{NCCP}(\alpha) = \frac{\alpha}{1-\alpha}$; this keeps griefing factors bounded between $\frac{1}{2}$ and 2.

## 4 Pools

In a traditional (ie. not sharded or otherwise scalable) blockchain, there is a limit to the number of validators that can be supported, because each validator imposes a substantial amount of overhead on the system. If we accept a maximum overhead of two consensus messages per second, and an epoch time of 1400 seconds, then this means that the system can handle 1400 validators (not 2800 because we need to count prepares and commits). Given that the number of individual users interested in staking will likely exceed 1400, this necessarily means that most users will participate through some kind of "stake pool".

There are several possible kinds of stake pools:

- **Fully centrally managed**: users $B_1 \ldots B_n$ send coins to pool operator $A$. $A$ makes a few deposit transactions containing their combined balances, fully controls the Prepare and Commit process, and occasionally withdraws one of their deposits to accommodate users wishing to withdraw their balances. Requires complete trust.
- **Centrally managed but trust-reduced**: users $B_1 \ldots B_n$ send coins to a pool contract. The contract sends a few deposit transactions containing their combined balances, assigning pool operator

$A$ control over the Prepare and Commit process, and the task of keeping track of withdrawal requests. $A$ occasionally withdraws one of their deposits to accommodate users wishing to withdraw their balances; the withdrawals go directly into the contract, which ensures each user's right to withdraw a proportional share. Users need to trust the operator not to get their deposits penalized, but the operator cannot steal the coins. The trust requirement can be reduced further if the pool operator themselves contributes a large portion of the coins, as this will disincentivize them from staking maliciously.

- **2-of-3**: a user makes a deposit transaction and specifies as validation code a 2-of-3 multisig, consisting of (i) the user's online key, (ii) the pool operator's online key, and (iii) the user's offline backup key. The need for two keys to sign off on a prepare, Commit or withdraw minimizes key theft risk, and a liveness failure on the pool side can be handled by the user using their backup key.
- **Multisig managed**: users $B_1 \dots B_n$ send coins to a pool contract that works in the exact same way as a centrally managed pool, except that a multisig of several semi-trusted parties needs to approve each Prepare and Commit message.
- **Collective**: users $B_1 \dots B_n$ send coins to a pool contract that that works in the exact same way as a centrally managed poolg , except that a threshold signature of at least portion $p$ of the users themselves (say, $p = 0.6$) needs to approve each Prepare and Commit messagge.

We expect pools of different types to emerge to accomodate smaller users. In the long term, techniques such as blockchain sharding will make it possible to increase the number of users that can validate directly, and extensions to allow validators to temporarily "drop out" from the validator set when they are offline can mitigate liveness risk.

## 5 Conclusions

The above analysis gives a parametrized scheme for incentivizing in Casper, and shows that it is a Nash equilibrium in an uncoordinated-choice model with a wide variety of settings. We then attempt to derive one possible set of specific values for the various parameters by starting from desired objectives, and choosing values that best meet the desired objectives. This analysis does not include non-economic attacks, as those are covered by other materials, and does not cover more advanced economic attacks, including extortion and discouragement attacks. We hope to see more research in these areas, as well as in the abstract theory of what considerations should be taken into account when designing reward and penalty schedules.

**Future Work.** [fill me in]

**Acknowledgements.** We thank Virgil Griffith for review.

## 6 References

## References

[1] V. Buterin. Minimal slashing conditions, 03 2017.

Optimal selfish mining strategies in Bitcoin; Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar: https://arxiv.org/pdf/1507.06183.pdf

Potential games; Dov Monderer and Lloyd Shapley: `http://econpapers.repec.org/article/eeegamebe/v\_3a14\_3ay\_3a1996\_3ai\_3a1\_3ap\_3a124-143.htm`

# Appendix

## 7 Unused text

[This is where text goes that for which a home hasn't been found yet. If no home is found, it will be deleted.]

Two other reasons to participate in stake pools are (i) to mitigate *key theft risk* (i.e. an attacker hacking into their online machine and stealing the key), and (ii) to mitigate *liveness risk*, the possibility that the validator node will go offline, perhaps because the operator does not have the time to manage a high-uptime setup.

[Do we want to require that the Prepare be done in the first 1/2 of the epoch? I'm mildly concerned there may not always be enough time to Commit.]

[Remember: The only block you're allowed to Prepare is the last block of each epoch.]

[Remember: Even if the Finalization goes through, the collective penalties are still applied.]