Programmable Privacy

Behold, the enigmatic programmable privacy! A concept cited far and wide, yet misunderstood like an elusive unicorn. Let us embark on a quest to shed light upon its true nature and discern the impacts and pitfalls that lurk beneath the surface.

We present a precise definition for programmable privacy after noticing that, despite community-wide citation and reference, there lacks an understanding of what programmable privacy actually is.

The only existing written definition is one from secret network. There are also some floating descriptions of what it is in the suave/mev-share post and notion/groupchat folklore, but none of them is satisfactory. Moreover, since the mentioning of programmable privacy as a concept in mev-share post, many teams have started abusing the term assuming that "it means something cool" despite nobody have a precise idea of it (and most people have a wrong idea of what it is).

Notations

- commitment type: $\mathcal{C}$
- commitment (inductive) constructors: K
- information space: I
- a subset of the information: I'
- strategy space (each strategy is a transaction): S
- outcome space (the ending EVM state): O
- searcher (users that can express preference on other user's information): s
- user: u
- generic user (combination of searcher and user): g

Informal Definition

Definition: programmable privacy means that agents have the ability to control the pathway of its information and how its information is used along the pathway.

- **Example**: I consent to giving my name to you, but only when this piece of information is used when registering driver's license, and it should expire after that.
- **Example**: I consent to giving my browser history to anyone that provides me with a piece of proof that the only context they can use the history in is to optimize for recommender systems.
- **Example**: I consent to giving my transaction data to anyone that has been proven to have no ability to extract frontrunning MEV based on my data.

Definition: a piece of information I is programmably private if the information, when used, can check the context of its usage and reject to be used if the context does not satisfy the preference of the information originator.

Informal Description

This section describes programmable privacy in terms of notions that have a structural correspondence with it.

Observation: note that the ability to control the pathway of how your information is used requires proof about the context. This context-awareness means that in programmable privacy, information, is treated as a resource and thus has a notion of usage.

- **Example**: consider the type system in languages like Move or Rust, where shared memory access (i.e., objects) is treated as a resource. To access such a resource, one must possess it (i.e., prove the correct usage of the resource), akin to locks and reference counts of locks in concurrent programming. In this context, the *usage of a term along the execution path* parallels the *usage of information along the execution path*. Consequently, the *well-typedness of the term* corresponds to the *fulfillment of programmable privacy requirements*.

- **Example**: continuing our type systems analogy, the treatment of information in programmable privacy corresponds to dynamic type systems (i.e., within runtime we check that the execution/usage of the information can only proceed if certain specifications/properties of the usage are satisfied).

- **Example**: in fact, many language/formal systems/specification (they are no different) design are just moving the hidden bits in context from the semantics of the environment to the semantics of the language and then to the syntax of the language. This can be clearly seen from the design of linear logic and relevance logic where the semantics (usage, revert or not) depends on the original syntax (context).

- **Example**: having programmable privacy is akin to achieving self-locating beliefs; e.g., in the sleeping beauty problem, the princess will be context-aware or a-priori-ly predicate her awakening on the date. This applies to the example of Newcomb's paradox as well, where a popular theory is that the paradox is caused by the inability to distinguish between reality or simulation (which is the consciousness being not context aware).

Remark: information wants to be free, it is memetic. Thus, it is hard to predicate/control the context of the information's usage, as:

- **Case 1**: Static control would necessitate the information owner to require input from all programs that use the information, including those that use the outputs of these programs, up to closure. And to determine whether these context programs satisfy the predicate for *allowed usage of the information*, one must slice and statically analyze them. However, static analysis proves ineffective in practice unless paired with economic incentives, such as builder-gas or PoW, to deter DoS attacks. Even with economic incentives, the game becomes overly complex, raising the barrier to entry (e.g., builder-gas golfing). In this scenario, people can write applications (deploy mechanisms) as static analyzer options, making programmable privacy enforcement more efficient.

- **Case 2**: Dynamic control hinders finality or fast confirmation guarantees. We cannot ensure the end state or verify successful enforcement of programmable privacy without executing everything, which inhibits our ability to provide timely confirmation.

Assumptions

- there exists a permissionless credible commitment device (PCCD) where one can post commitments and those commitments will be executed faithfully. Examples of such device include SGX, committee-based (rollup, sidechain, validium, blockchain), MPC, zkp, etc,.

- we use the word "certainty" to mean cases where there is no "fundamental uncertainty" (for an elaboration see last section "On Uncertainty")

First case: certainty

If we assume there is full certainty, i.e., the context of the usage of the information can be fully (computationally-feasibly) controlled/predicated on/determined/measured, then we are in this case. We give some examples.

Searchers utilizes full information I the searcher makes a commitment that takes in a user commitment and returns a commitment to the usage of the information, i.e., its context; the user makes a commitment that takes in the searcher's commitment and returns a commitment to change its' transaction content based on how the searcher will use its transaction (e.g., change the transaction to instant revert if none of the usage is satisfactory).

- $K_s: (\mathcal{C}_u \rightarrow S_s) \rightarrow \mathcal{C}_s$

- $K_u: (\mathcal{C}_s \rightarrow S_u) \rightarrow \mathcal{C}_u$

Searchers utilizes full information I with bonds: the searcher makes a commitment that takes in a user commitment and returns a commitment to how it will make transfers given the outcome (e.g., transferring the user 2 ETH if there existed another transaction in front of user's transaction); the user makes a commitment that takes in the searcher's commitment (to transfers) and returns a commitment to change its' transaction content based on how the searcher will use its transaction.

- $K_s: (\mathcal{C}_u \rightarrow (O \rightarrow \mathbb{R})) \rightarrow \mathcal{C}_s$
- $K_u: (\mathcal{C}_s \rightarrow S_u) \rightarrow \mathcal{C}_u$

Obviously if we have full certainty, assuming a well-defined semantics for resolving commitments, best outcome is always achieved (i.e., assuming all agents are rational equilibria in commitment game is max welfare in original game).

Second case: uncertainty

Uncertainties are way more common in reality due to the computational hardness of enforcing context-aware predicates. Obviously without full certainty we cannot possibly achieve maximal welfare, so privacy predicates like $I' \subseteq I$ here acts as overapproximations (i.e., they are discrete steps we take to approximate some context predicate in the full certainty case). We give some examples.

Searchers make transfers on partial information: the searcher makes a commitment that takes in a user commitment and returns a transfer commitment (e.g., if you release the amount of the swap I'll transfer you 2 ETH); the user makes a commitment that takes in the searcher's commitment and returns a commitment to release certain part of the transaction's information (e.g., direction of swap).

- $K_s: (\mathcal{C}_u \rightarrow \mathbb{R}) \rightarrow \mathcal{C}_s$
- $K_u: (\mathcal{C}_s \rightarrow (I' \subseteq I)) \rightarrow \mathcal{C}_u$

Searchers make slashable actions on partial information: the searcher makes a commitment that takes in a user commitment and returns a commitment that makes transfers based on the outcome state after execution (e.g., searcher promises to not give user a slippage greater than 1% given that user is willing to release the amount of the swap, and if the searcher was found to give a greater slippage, it transfers user 1 ETH); the user makes a commitment that takes in the searcher's commitment and returns a commitment to release certain part of the transaction's information.

- $K_s: (\mathcal{C}_u \rightarrow (O \rightarrow \mathbb{R})) \rightarrow \mathcal{C}_s$
- $K_u: (\mathcal{C}_s \rightarrow (I' \subseteq I)) \rightarrow \mathcal{C}_u$

Common Variants

We present some variants of programmable privacy that is often discussed or mis-treated as the full version of programmable privacy.

Users can control which information to release: users can choose which part of their information gets revealed to the searcher. Users cannot condition their choice of revelation based on the context, and searchers are assumed to have the ability to take unconstrained actions.

- $K_s: \mathcal{C}_s$
- $K_u: (I' \subseteq I) \rightarrow \mathcal{C}_u$

This is the programmable privacy notion mentioned in most posts.

Users can control which information to release given prior bonded commitments by searchers on usage users can choose which part of their information gets revealed to the searcher based on what the searcher's commitment is, and searchers commit to a transfer that predicates on the ending state (e.g., if in the outcome the searcher was found to be frontrunning, then the searcher transfers `0x0` the bond that it put up before).

- $K_s: (O \rightarrow \mathbb{R}) \rightarrow \mathcal{C}_s$

- $K_u: (\mathcal{C}_s \rightarrow (I' \subseteq I)) \rightarrow \mathcal{C}_u$

Adverse selection and higher-order commitments

In those the variant cases, commitments made by searchers and users ($\mathcal{C}_u$ and $\mathcal{C}_s$) are not conditioned on each other, which means there is the problem of adverse selection (free options) which introduce latency race situations.

This problem was partially solved in the previous cases where commitments are allows to condition on each other (as we are giving power to the maker to change its commitment given what the taker commitment is). And this interdependency makes the commitments higher-order.

However, the problem that the maker commitment was made in time t instead of time $t+1$ still gives the maker a disadvantage. We can mitigate this disadvantage using MEV-time Oracles (i.e., have the commitment condition on the information in time $t+1$ even though the commitment was made on time t), e.g., commitment depends on real-time binance price which will be provided as oracle by the PCCD provider/validator.

Multiple players and semantics

If we generalize the commitment constructors to N users and M searchers, we get:

- $K_s: (\mathcal{C}*{u \in N} \rightarrow \mathcal{C}*{s' \in M \setminus s} \rightarrow S_s) \rightarrow \mathcal{C}_s$
- $K_u: (\mathcal{C}*{s \in M} \rightarrow \mathcal{C}*{u' \in N \setminus u} \rightarrow S_u) \rightarrow \mathcal{C}_u$

or:

- $K_s: (\mathcal{C}*{u \in N} \rightarrow \mathcal{C}*{s' \in M \setminus s} \rightarrow (O \rightarrow \mathbb{R})) \rightarrow \mathcal{C}_s$
- $K_u: (\mathcal{C}*{s \in M} \rightarrow \mathcal{C}*{u' \in N \setminus u} \rightarrow (I' \subseteq I)) \rightarrow \mathcal{C}_u$

If we generalize the commitment constructors to N generalized users g (a combination of users and searchers), we get:

- $K_g: (\mathcal{C}_{g' \in N \setminus g} \rightarrow S_g) \rightarrow \mathcal{C}_g$

Semantics: This is essentially *smart transactions* and *program equilibria* for which there exists many (but not enough in the author's opinion) literature guiding us on how to implement the semantics for those commitments.

Of course, one worry is that when resolving those higher-order commitments, there exists some privacy problem. For example, one could easily frontrun others' transaction if the other person agrees to being frontran. The problem of "can voluntary actions take place despite socially we feel it's not to be encouraged" is an interesting question.

Another example is what if the user's commitment includes a high value whitehat transaction and the searcher's commitment is one that resembles a generalized frontrunner. Here, if wanted, we can add extra privacy without loss of efficiency by not predicating on the commitment but on an execution function that always gives all the possible outcomes given others' commitments.

Another worry is non-termination: many of the previous definition we gave look suspicously like Curry's paradox. Indeed, they are. This implies that malicious actors can send commitments that are intentionally non-terminating when faced with other commitments, thus breaking our programmable privacy enforcement platform if the semantics of those commitments are not well-defined such that it's robust to attacks. This can be mitigated with the addition of builder-gas and templates/applications.

Cognitive Burden, Applications, and Decentralized Use of Knowledge

A significant challenge with higher-order commitments in programmable privacy is the mental burden placed on users. We

can alleviate this issue by deploying templates that outline the commitment structure, as these templates act like privacy applications (akin to Uniswap being an application). With permissionless deployment of templates, anyone can be a "privacy mechanism designer."

For example, one could deploy a commitment template that allows sharing of swap directions in exchange of a guaranteed better execution (than if the direction was not revealed). So when users are constructing commitments, they are most likely to choose among existing applications/templates. This is similar to how, on Ethereum, one can choose to send a transaction that is a custom re-org contract to attack the economic stability, but most transaction flow is still gonna be just constructed from templates of applications (e.g., a call to `collectAllFees` on Uniswap). Essentially, users would likely choose from existing templates when constructing commitments, much like Ethereum transactions rely on templates from applications.

Assuming there is enough incentive (which they have, as there can be MEV capture) for people to deploy commitment templates (applications on our programmable privacy engine/PCCD), we will reach a state where most commitments can efficiently be resolved (terminate), solving both the cognitive and computational burden problem.

One might be tempted to ask: "if we are doing templates then what is the point of having a higher-order commitment constructor that allows possible non-termination?" The reason is similar to why Ethereum supports a pseudo-Turing-complete virtual machine: we want permissionless deployment of applications and do not want to restrict what kind of applications/templates/mechanisms can be deployed (such that the decentralized knowledge in society can be maximally utilized).

The cognitive burden can also be lowered by the recent rise of AI (e.g., ask GPT-4 to encode your natural language commitment/preference into a smart transaction and have Ethereum/SUAVE semantics settle those transactions in a credible way).

Conclusion

Programmable Privacy is the idea that information can control its usage. This idea can be realized through higher-order commitments (smart transactions) on a permissionless credible commitment device.

Implementing programmable privacy in this manner ensures maximal expressiveness, which in turn leads to optimal efficiency and enhanced welfare. This is because conditioning on a wider range of factors increases certainty, reducing risk and improving efficiency. Additionally, it helps mitigate adverse selection problems associated with free options.

Nevertheless, employing higher-order commitments introduces challenges, such as non-terminating semantics, high cognitive burden, and computational intensity. We have demonstrated that the use of templates/applications, which are permissionless to deploy, can effectively mitigate these issues.

On Uncertainty

Expanding on the taxonomy of different uncertainties, the elimination of strategic and informational uncertainties via commitments require the information and strategy/action space to be able to be predicated on and determined without ambiguity. As a result, they must be known unknowns (that can become known knowns).

Of course, in reality, even if we can predicate on the action space, the cost of computing/determining the validity of that predicate is often too high, so some actions/information might as well be modeled as fundamentally uncertain. Therefore, predicating on them loses purpose. Moreover, our physical world limits us to what information can be measured/determined so there are fundamentally some predicates that we cannot use, such as time-based ordering of transactions in distributed systems.

Those uncertainties, despite fundamental, are all known unknowns (and we know they cannot be known, ever), and we can simply not have our commitments predicate on them. On the other hand, the existence of unknown unknowns (unknown uncertainties) poses a greater threat to commitments: because they are truly unknown, we cannot possibly predicate on them or not predicate on them.

As a result, when unknown unknowns becomes known, they can invalidate many commitments. Furthermore, we cannot possibly account for unknown unknowns when making commitments while keeping the commitments credible, as being

credible would mean our change to the commitment is able to be predicated on.

Therefore, we use the term fundamental uncertainty (to contrast the predicatable uncertainties that PCCDs can deal with, namely informational and strategic) to refer the three cases of limitations as to coordination using commitments:

- (i) the unknown unknowns
- (ii) the known unknowables
- (iii) the known knowables that cannot be efficiently determined/computed.

We probably live in a world where all those three kinds of fundamental uncertainties are prevalent. As a result, the efficiency boost we can possibly achieve though the use of PCCDs is limited by the boundary of those uncertainties.

An interesting side note is that we can often exchange different kinds of uncertainties with each other: the fundamental uncertainty in one dimension might be transformed into some more tractable uncertainty that is predicatable in another dimension.

For example, if it is hard to measure the outcome of some game (e.g., a distrust of oracles, or simply because the space is too huge), then maybe we can try to determine the certainty of player's action space by delegating game play to algorithms.

Another effect of the existence of fundamental uncertainties is that they turn one's position of being the first one to commit (the Stackelberg leader) from a blessing into a curse, because the longer one waits to fix the content of the commitment, the less fundamental uncertainties there are. So the one who commits later might actually face less uncertainty (risk) than others who commit earlier.

Essentially, this is the case where commitment "takers" get a free option as they "snipe" the stale quotes from the "maker" of commitments.

Sometimes this elimination of fundamental uncertainty for the one who commits later gives so much advantage such that being the Stackelberg leader becomes an unfavorable position because the gains from elimination of strategic uncertainty and informational uncertainty is less than the gain from elimination of fundamental uncertainty.