How to Use Bitcoin to Incentivize Correct Computations

Ranjit Kumaresan
Technion

Iddo Bentov
Technion

CCS 2014

Formal model that incorporates coins

Ideal functionalities $\mathcal{F}^{\star}_{\square}$ with coins

- If party P_i has (say) secret key sk_i and sends it to party P_j, then both P_i and P_j will have the string sk_i.
- If party P_i has coins(x) and sends $y < x$ coins to party P_j, then P_i will have have coins$(x - y)$ and P_j will coins(y).

Ideally, all the parties deem coins to be valuable assets.

We define *Secure computation with coins* and provide proofs using the simulation paradigm (but not in this talk).

Claim-or-Refund for two parties P_s,P_r (implicit in [Max11],[BBSU])

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ Claim-or-Refund ideal functionality

1. The sender P_s deposits (locks) her coins(q) while specifying a timebound τ and a circuit $\phi(\cdot)$.
2. The receiver P_r can claim (gain possession) of the coins(q) by publicly revealing a witness w that satisfies $\phi(w) = 1$.
3. If P_r didn't claim within time τ, coins(q) are refunded to P_s.

How to realize $\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin

- The feature that is needed is "timelock" transactions.
- Technically: Bitcoin nodes agree to include a transaction with timelock field τ only if current block index/timestamp is $> \tau$
- It is possible to have more expressive schemes that allow not-yet-reached timelock transactions to reside on the blockchain (or local mempool), but this is prone to DoS.

$\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin

High-level description the $\mathcal{F}^{\star}_{\mathrm{CR}}$ implementation in Bitcoin

- P_s controls TX_{old} that resides on the blockchain.
- P_s creates a transaction TX_{new} that spends TX_{old} to a Bitcoin script that can be redeemed by P_s and P_r, or only by P_r by supplying a witness w that satisfies $\phi(w) = 1$.
- P_s asks P_r to sign a timelock transaction that refunds TX_{new} to P_s at time τ (conditioned upon both P_s and P_r signing).
- After P_r signs the refund, P_s can safely broadcast TX_{new}.

1. P_s is safe because P_r only sees $\texttt{hash}(TX_{\mathsf{new}})$, and therefore cannot broadcast TX_{new} to cause P_s to lose the coins.
2. P_r can safely sign because the protocol uses freshly generated (sk_R, pk_R) pair.

The structure of Bitcoin transactions

How standard Bitcoin transaction are chained

- TX_{old} = earlier TX output of coins(q) is redeemable by pk_A
- $id_{\text{old}} = \texttt{hash}(TX_{\text{old}})$
- $PREPARE_{\text{new}} = (id_{\text{old}}, q, pk_B, 0)$ — 0 means no timelock
- $TX_{\text{new}} = (PREPARE_{\text{new}}, \texttt{Sign}_{sk_A}(PREPARE_{\text{new}}))$
- $id_{\text{new}} = \texttt{hash}(TX_{\text{new}})$
- Initial minting transaction specifies some pk_M that belongs to a miner, and is created via *proof of work*.

Realization of $\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction

- $PREPARE_{\textsf{new}} = (id_{\textsf{old}}, q, (pk_S \wedge pk_R) \vee (\phi(\cdot) \wedge pk_R), 0)$
- $\phi(\cdot)$ can be $\texttt{SHA256}(\cdot) == Y$ where Y is hardcoded.
- $TX_{\textsf{new}} = (PREPARE_{\textsf{new}},\ \texttt{Sign}_{sk_S}(PREPARE_{\textsf{new}}))$
- $id_{\textsf{new}} = \texttt{hash}(TX_{\textsf{new}})$
- P_s sends $PREPARE_{\textsf{refund}} = (id_{\textsf{new}}, q, pk_S, \tau)$ to P_r
- P_r sends $\sigma_R = \texttt{Sign}_{sk_R}(PREPARE_{\textsf{refund}})$ to P_s
- P_s broadcasts $TX_{\textsf{new}}$ to the Bitcoin network
- If P_r doesn't reveal w until time τ then P_s creates $TX_{\textsf{refund}} = (PREPARE_{\textsf{refund}},\ (\texttt{Sign}_{sk_S}(PREPARE_{\textsf{refund}}), \sigma_R))$ and broadcasts it to reclaim her q coins

Fairness with penalties

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn't deliver output to honest parties $\Rightarrow$ every honest party is compensated

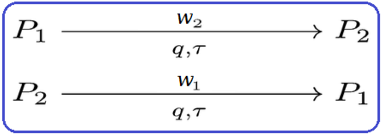
Outline of $\mathcal{F}_f^\star$ – fairness with penalties for any function f

- $P_1, \ldots, P_n$ run secure *unfair* MPC for $f(x_1, \ldots, x_n)$ that
  1. Computes shares $(y_1, \ldots, y_n)$ of the output $y = f(x_1, \ldots, x_n)$
  2. Computes $\mathrm{Tags} = (\mathtt{com}(y_1), \ldots, \mathtt{com}(y_n))$ $= (\mathtt{hash}(y_1), \ldots, \mathtt{hash}(y_n))$
  3. Delivers (y_i, Tags) to every P_i
- $P_1, \ldots, P_n$ deposit coins and run fair reconstruction (fair exchange) with penalties to swap the y_i's among themselves.

Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction

"Abort" attack:

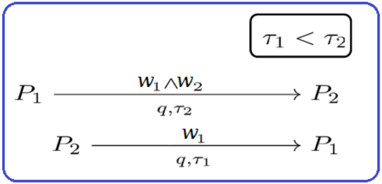
P_2 claims without deposting



Fair exchange:

P_1 claims by revealing w_1

$\Rightarrow$ P_2 can claim by revealing w_2

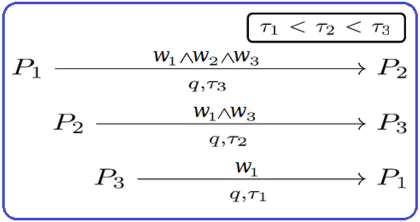


Malicious coalition:

Coalition P_1, P_2 obtain w_3 from P_3

P_2 doesn't claim the top transaction

P_3 isn't compensated



Incentivizing delegated computation

- Let f be is a function with high complexity.
- Delegator D wishes to pay worker W to compute $y = f(u)$.
- Assume that the Bitcoin scripting language is Turning complete, or can otherwise compute $f(u)$ given input u.
  - Current Bitcoin scripts are purposely not Turning complete, and have bounded size (which implies fast running time).
  - Requiring higher fees for more complex scripts isn't so simple, because network DoS risks mean that nodes who propagate the transaction (without getting paid) must verify the script first.
  - Projects like Ethereum wish to support Turing complete scripts, where the user pays a fee for a quota of script steps and if the quota runs out then it needs to be refilled, but maybe a supplemental flat fee is also needed to avoid DoS, so if (?!) it can work then users will need to pay higher fees relative to Bitcoin.

Incentivizing delegated computation (contd.)

What we don't want to do

- D sends (f, u) to W.
- D creates $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction with circuit $\phi_{f,u}(\cdot)$ that lets W redeem coins(q) if W reveals w such that $\phi_{f,u}(w) = 1$, where the circuit/script $\phi_{f,u}(w)$ is satisfied iff $f(u) = w$.

$\Rightarrow$ All nodes in the Bitcoin network need to compute $f(u)$ when validating this script, while only W gets paid.

Note: why the need to have a specific worker W in this scheme?

- $\mathcal{F}^{\star}_{\mathrm{CR}}$ as specified is a 2-party protocol.
- But the problem is inherent, see bounties...

Definitions of verifiable computation (with trusted setup)

Public verifiable computation scheme

- $(ek_f, vk_f) \leftarrow \text{KeyGen}(f, 1^\lambda)$: randomized keygen algorithm that takes the function f to be outsourced and security parameter λ; it outputs a public evaluation key ek_f, and a public verification key vk_f.
- $(y, \psi_y) \leftarrow \text{Compute}(ek_f, u)$: deterministic worker algorithm that uses ek_f and the input u to output $y \leftarrow f(u)$ and a proof ψ_y of y's correctness.
- $\{0,1\} \leftarrow \text{Verify}(vk_f, u, (y, \psi_y))$: deterministic verification algorithm that uses vk_f with input u and witness (y, ψ_y) to output 1 iff $f(u) = y$.

Efficiency KeyGen is assumed to be a one-time operation whose cost is amortized over many calculations, and Verify is cheaper than evaluating f.

Correctness
$$\Pr\left[\begin{array}{l}(ek_f, vk_f) \leftarrow \text{KeyGen}(f, 1^\lambda), (y, \psi_y) \leftarrow \text{Compute}(ek_f, u) : \\ 1 = \text{Verify}(vk_f, u, (y, \psi_y))\end{array}\right] = 1$$

Soundness For any PPT adversary $\mathcal{A}$ the following is negligible in λ:
$$\Pr\left[\ (u', y', \psi'_y) \leftarrow \mathcal{A}(ek_f, vk_f) : f(u') \neq y' \wedge 1 = \text{Verify}(vk_f, u', (y', \psi'_y))\ \right]$$

$\mathcal{F}^{\star}_{\text{exitCR}}$

- Reminder: the script of $\mathcal{F}^{\star}_{\text{CR}}$ is $\boxed{(pk_S \wedge pk_R) \vee (\phi(\cdot) \wedge pk_R)}$
- The timelocked refund transaction $(id_{\text{new}}, q, pk_S, \tau)$ is done by satisfying the condition $\boxed{(pk_S \wedge pk_R)}$
- Hence P_s and P_r can also decide to sign a non-timelocked transaction $(id_{\text{new}}, q, pk_R, 0)$ that will satisfy this same condition and transfer the coins(q) to P_r, so we can easily get:

The ideal functionality $\mathcal{F}^{\star}_{\text{exitCR}}$

1. The sender P_s deposits (locks) her coins(q) while specifying a timebound τ and a circuit $\phi(\cdot)$.
2. The receiver P_r can claim (gain possession) of the coins(q) by publicly revealing a witness w that satisfies $\phi(w) = 1$.
3. At any point before time τ, P_s and P_r can agree to release the coins(q) to P_r without revealing w.
4. If P_r didn't claim within time τ, coins(q) are refunded to P_s.

Correct scheme for incentivized delegated computation

Non-private delegated computation

1. D and W engage in secure computation to obtain $(ek_f, vk_f) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$.
2. D sends u to W.
3. D and W invoke $\mathcal{F}^{\star}_{\text{exitCR}}$ with circuit $\phi(\cdot) = \mathsf{Verify}(vk_f, u, \cdot)$ and timebound τ that lets W earn D's coins(q) if W reveals $w = (y, \psi_y)$ such that $\mathsf{Verify}(vk_f, u, (y, \psi_y)) = 1$.
4. W executes $(y, \psi_y) \leftarrow \mathsf{Compute}(ek_f, u)$ and sends $w = (y, \psi_y)$ to D within time $\tau' < \tau$.
5. D verifies (y, ψ_y), then D and W release the coins(q) to W.
6. If D doesn't release the coins(q) until time τ' then W will redeem the $\mathcal{F}^{\star}_{\text{exitCR}}$ transaction between time τ' and τ and claim the coins(q).

Correct scheme for incentivized delegated computation (contd.)

Due to having a trusted setup...

- The work done by D to compute KeyGen$(f, 1^\lambda)$ will be amortized over multiple executions for $f(u_1), f(u_2), \ldots$
- If D invoked KeyGen$(f, 1^\lambda)$ herself then a malicious D can cause an honest W to do work without getting paid.

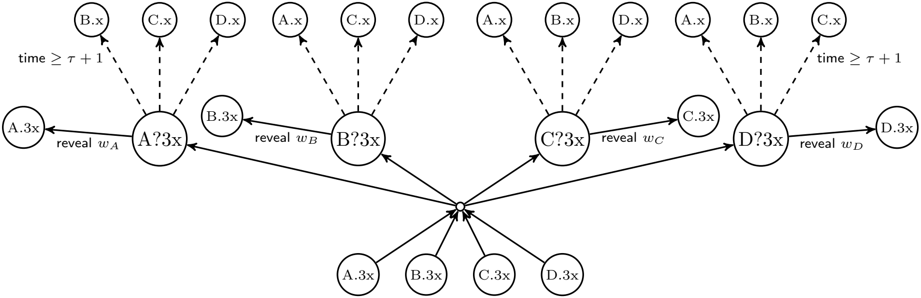
- If D is honest or rational then she will release the coins(q) to W upon receiving $w = (y, \psi_y)$ before time τ', because W can claim the coins(q) until time τ anyway.

$\Rightarrow$ unless D is purely malicious, all the Bitcoin nodes will validate ordinary ECDSA signatures rather than evaluate Verify$(vk_f, u, \cdot)$, which is an order of magnitude faster.

Incentivizing private delegated computation

- The previous scheme makes u and $f(u)$ public.
- It is possible to use a private verification scheme (employing homomorphic encryption).
- Note: if W knows vk_f then W can cheat.
- P_s and P_r will run secure computation also for Verify(vk_f, u, w) where W's input is $w = (y, \psi_y)$, which will secret share y between P_s and P_r if ψ_y is a correct proof.
- Then P_s,P_r will invoke $\mathcal{F}^{\star}_{\mathrm{CR}}$ to pay P_r if she reveals her share.
- Full scheme also needs to ensure that inputs are consistent across the secure computations of KeyGen and Verify.
- We can avoid "rejection" attack of supplying incorrect proof to learn information, by letting W create $\mathcal{F}^{\star}_{\mathrm{CR}}$ that pays to P_s if she reveals the right output, and invoking secure MPC with penalties that delivers the right output to P_s iff W supplies an incorrect proof (else P_s gets commitment to the right output).
- Honest W isn't guaranteed payment (due to trusted setup...)

Multilock



In principle, jointly locking coins for fair exchange can work well:

1. $M =$ "if P_1, P_2, P_3, P_4 sign this message with inputs of coins(x) each then their $3x$ coins are locked into 4 outputs of coins(x) each, where each P_i can redeem output T_i with a witness w_i that satisfies ϕ_i, and after time τ anyone can divide an unredeemed output T_i equally to $\{P_1, P_2, P_3, P_4\} \setminus \{P_i\}$"
2. P_1, P_2, P_3, P_4 sign M and broadcast it, and after M is confirmed, each P_i redeems coins(x) by revealing w_i

The multilock functionality $\mathcal{F}^{\star}_{\mathrm{ML}}$

Hence the functionality $\mathcal{F}^{\star}_{\mathrm{ML}}$ holds the following attributes:

- The atomic nature of $\mathcal{F}^{\star}_{\mathrm{ML}}$ guarantees that either all the n parties agreed on the circuits $\phi_i(\cdot)$, the timebound τ, and the security deposit amount x, or else no coins become locked.
- Each corrupt party who aborts after the coins become locked is forced to pay coins$(\frac{x}{n-1})$ to each honest party.
- If P_i reveals w_i s.t. $\phi_i(w_i) = 1$ then w_i becomes public.
- The limit τ prevents the possibility that a corrupt party learns the witness of an honest party, and then waits for an indefinite amount of time before recovering her own coins amount.

We prove using the simulation paradigm that $\mathcal{F}^{\star}_{f}$ can be securely computed with penalties in the $\mathcal{F}^{\star}_{\mathrm{ML}}$-hybrid model.

$\mathcal{F}^{\star}_{\mathrm{ML}}$ and Bitcoin

- $\mathcal{F}^{\star}_{\mathrm{CR}}$ is asymmetric: P_s prepares the complete transaction, then sends only its hash so that P_r can sign the refund before locking P_s's coins.
- $\mathcal{F}^{\star}_{\mathrm{ML}}$ is symmetric: if a corrupt P_i obtains the complete transaction that locks everyones coins before the refunds are done, then P_i can cause honest parties to lose coins.
- Reminder: $PREPARE_{\text{new}} = (id_{\text{old}}, q, pk_B, 0)$
- $TX_{\text{new}} = (PREPARE_{\text{new}}, \mathtt{Sign}_{sk_A}(PREPARE_{\text{new}}))$, $\boxed{id_{\text{new}} = \mathtt{hash}(TX_{\text{new}})}$
- If we have $id_{\text{new}} = \mathtt{hash}(PREPARE_{\text{new}})$ then parties can reference unsigned transactions when creating the refunds.
- This also enables richer forms of contracts: if P_1 can redeem a transaction to P_2 in two separate ways, then P_2 can create a future transaction that redeems coins to P_3 only if P_1 operated in a certain way.

$\mathcal{F}^{\star}_{\mathrm{ML}}$ and Bitcoin (contd.)

- There is also a disadvantage: if P_1 can redeem by revealing either of two witnesses w, w', and we reference her transaction via id_{new} hash that doesn't express which witness was revealed, then a contract that relies on (say) w' being revealed cannot rely on the Bitcoin blockchain to provide this evidence.

Our Bitcoin enhancement proposal

- Reference previous txid via $id^{\text{simp}}_{\text{new}} = \mathtt{hash}(PREPARE_{\text{new}})$.
- Use $id_{\text{new}} = \mathtt{hash}(TX_{\text{new}})$ for the leaves of the Merkle root that gets committed via *Proof of Work*.

- This enables the best of both worlds (including $\mathcal{F}^{\star}_{\mathrm{ML}}$).
- No extra `hash` invocation required because $id^{\text{simp}}_{\text{new}}$ must be computed anyway before signing the incomplete transaction.

Summary of multiparty fair exchange via Bitcoin

- $\mathcal{F}^{\star}_{\mathrm{ML}}$ requires $\mathcal{O}(1)$ Bitcoin rounds and $\mathcal{O}(n^2)$ transaction data (and $\mathcal{O}(n^2)$ signature operations), while the ladder requires $\mathcal{O}(n)$ Bitcoin rounds and $\mathcal{O}(n)$ transactions.

Recap:

- Multiparty fair computation can be implemented in Bitcoin via the ladder construction.
- Multiparty fair computation can be implemented via $\mathcal{F}^{\star}_{\mathrm{ML}}$ with an enhanced Bitcoin protocol.

The DualEx 2-party secure computation protocol [MF06, HKE12]

- Secure computation in the malicious setting, which commonly relies on cut-and-choose or ZK proofs, is less efficient than in the semihonest setting.

Observation

2-party semihonest secure computation by Yao's garbled circuit protocol preserves **privacy** against a malicious circuit generator as long as:

1. OT protocol that is secure against active attacks is used for the input wires.
2. The circuit evaluator doesn't reveal her output to the circuit generator.

The DualEx 2-party secure computation protocol (contd.)

- The DualEx protocol operates as follows:
  1. Execute Yao's protocol with P_1 as the circuit generator so that only the circuit evaluator P_2 obtains output.
  2. Execute Yao's protocol again with swapped roles.
  3. Test equality by using a protocol that is secure in the malicious setting to compare the outputs.

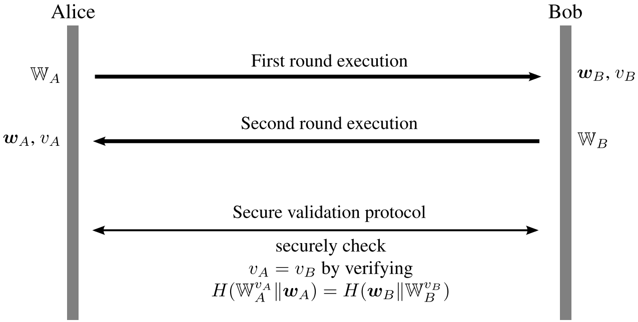


Figure 1. DualEx protocol overview (informal).

DualEx with penalties

- The equality test leaks a single-bit predicate to the adversarial party $\Rightarrow$ the adversary $\mathcal{A}$ learns a single bit on average.
  - On one extreme $\mathcal{A}$ can choose to learn a single bit always.
  - On the other extreme $\mathcal{A}$ can choose to learn whether the entire input of the honest party is a specific value.
- **Whenever the equality test fails $\mathcal{A}$ is caught, so our goal is to force $\mathcal{A}$ to pay coins to the honest party in this case.**

The core idea

If the equality test fails, then each party can claim coins (that the other party deposited) if she can produce **ZK** proof that she sent a correct garbled circuit when she acted as the circuit generator.

- The complexity of this ZK proof depends on $|f|$, and cannot be verified by using the current Bitcoin scripting language.

DualEx with penalties: obstacle 1/4

- **Obstacle #1:** The corrupt party should be able to claim only if the equality test failed, otherwise if the honest party deposit first then she can lose coins.

Remedy #1

- Each party privately generates auxiliary random data.
- Rewards can only be claimed if this random data is provided.
- By using secure computation, this data is released to both parties only if the equality test fails.

DualEx with penalties: obstacle 2/4

- **Obstacle #2:** The corrupt party can try to learn information by providing output keys that are junk or inconsistent with the correct garbled circuit that she constructed.

Remedy #2

- We derive a garbled circuit from a random seed.
- Then enforce that each party P_i must use the same seed ω_i for the committed garbled circuit and the output wires.
- This means that the output wires used in the garbled circuit and in the equality test are the same.
- The enforcement is done by a protocol that is secure against active attacks, which is also the case for the plain DualEx.

DualEx with penalties: obstacle 3/4

- **Obstacle #3:** The corrupt party can abort upon being caught, before the honest party obtains the auxilary data that she needs in order to provide ZK proof that she constructed a correct garbled circuit and obtain her reward.

Remedy #3

- We use fair secure computation with penalties for the equality test, hence the honest party will gain even more coins from the corrupt party if the corrupt party aborts.
- Side-effect: now we also guarantee fairness (with penalties), i.e., output delivery to both parties, unlike the plain DualEx.

DualEx with penalties: obstacle 4/4

- **Obstacle #4:** The corrupt party can cheat in other ways too, namely by providing inconsistent inputs in different stages, or causing selective failures with the inputs that she provides.

Remedy #4

- We deploy a secure computation protocol against active attacks for the sending the input wires too (instead of just using OT).
- This computations makes sure that the seed that derives the input wires is the same as the seed that derives the committed garbled circuit $\Rightarrow$ this garbled circuit has the same input wires.

- We prove security against all possible attacks via simulation.

The full protocol for DualEx with penalties

Input from P_1: m, x_1, ω_1.
Input from P_2: m, x_2, ω_2.

Output to both P_1 and P_2:

- Create $\mathbb{U}_1 \leftarrow \mathsf{iGb}(1^\lambda, \omega_1, m)$ and $\mathbb{U}_2 \leftarrow \mathsf{iGb}(1^\lambda, \omega_2, m)$.
- Compute $g_1' = \mathsf{com}(\omega_1; \rho_1)$ and $g_2' = \mathsf{com}(\omega_2; \rho_2)$ where ρ_1, ρ_2 are picked uniformly at random.
- Output $(\mathbb{U}_2^{x_1\|x_2}, g_2', \rho_1)$ to P_1 and $(\mathbb{U}_1^{x_1\|x_2}, g_1', \rho_2)$ to P_2.

Figure 1: Secure key transfer subroutine KT**.**

Input from P_1: $\ell_1 = \ell, \mathbf{w}_1, \omega_1, \rho_1, r_1, h_2, g_2'$.
Input from P_2: $\ell_2 = \ell, \mathbf{w}_2, \omega_2, \rho_2, r_2, h_1, g_1'$.

Output to both P_1 and P_2:

- If $\ell_1 \neq \ell_2$ or $\mathsf{H}(r_1) \neq h_1$ or $\mathsf{H}(r_2) \neq h_2$ or $\mathsf{com}(\omega_1; \rho_1) \neq g_1'$ or $\mathsf{com}(\omega_2; \rho_2) \neq g_2'$, output bad and terminate.
- Create $\mathbb{W}_1 \leftarrow \mathsf{oGb}(1^\lambda, \omega_1, \ell)$ and $\mathbb{W}_2 \leftarrow \mathsf{oGb}(1^\lambda, \omega_2, \ell)$.
- Check if $\exists v_1, v_2 \in \{0,1\}^\ell$ such that $\mathbb{W}_1^{v_1} = \mathbf{w}_2$ and $\mathbb{W}_2^{v_2} = \mathbf{w}_1$. If the check fails output bad and termiante.
- Check if $\exists v \in \{0,1\}^\ell$ such that $\mathbb{W}_1^{v} = \mathbf{w}_2$ and $\mathbb{W}_2^{v} = \mathbf{w}_1$. If check fails output (r_1, r_2). Else, output v.

Figure 2: Secure equality validation subroutine SV**.**

Inputs: P_1, P_2 respectively hold inputs $x_1, x_2 \in \{0,1\}^m$.
Preliminaries. Let $(\mathsf{com}, \mathsf{dec})$ be a perfectly binding commitment scheme. Let NP language $\mathcal{L}$ be such that $u = (a, b) \in \mathcal{L}$ iff there exists α, β such that $a = \mathsf{Gb}(1^\lambda, f, \alpha)$ and $b = \mathsf{com}(\alpha; \beta)$. Let $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a non-interactive zero knowledge scheme for $\mathcal{L}$. Let $\mathsf{crs} \leftarrow \mathcal{K}(1^\lambda)$ denote the common reference string. Let H be a collision-resistant hash function.
Protocol: For each $i \in \{1,2\}$, P_i does the following: Let $j \in \{1,2\}, j \neq i$.

1. Pick ω_i at random and compute $G_i \leftarrow \mathsf{Gb}(1^\lambda, f, \omega_i)$.
2. Send $(\mathsf{input}, sid, ssid, (m, x_i, \omega_i))$ to $\mathcal{F}_{\mathsf{KT}}$. If the output from $\mathcal{F}_{\mathsf{KT}}$ is abort, terminate. Else let output equal $(\mathbb{U}_j', g_j')$.
3. Send G_i to P_j and receive G_j from P_j.
4. Compute $\mathbf{w}_i \leftarrow \mathsf{Eval}(G_j, \mathbb{U}_j')$.
5. Choose random r_i and send $h_i = \mathsf{H}(r_i)$ to P_j.
6. Let $X_i = (G_j, g_j', h_j)$, and let $\phi_i(w; X_i) = 1$ iff $w = (\alpha, \beta)$ such that $\mathcal{V}(\mathsf{crs}, (G_i, g_i'), \alpha) = 1$ and $\mathsf{H}(\beta) = h_j$. Send $(\mathsf{deposit}, sid, ssid, i, j, \phi_i(\cdot; X_i), \tau, \mathsf{coins}(q))$ to $\mathcal{F}^\star_{\mathrm{CR}}$.
7. If no corresponding deposit message was received from $\mathcal{F}^\star_{\mathrm{CR}}$ on behalf of P_j, then wait until round $\tau + 1$ to receive refund message from $\mathcal{F}^\star_{\mathrm{CR}}$ and terminate.
8. Send $(\mathsf{input}, sid, ssid, (\ell_i, \mathbf{w}_i, \omega_i, r_i, h_j, g_j'), \mathsf{coins}(d))$ to $\mathcal{F}^\star_{\mathsf{SV}}$. Let z_i denote the output received from $\mathcal{F}^\star_{\mathsf{SV}}$. Do: (1) If $z_i = \perp$, then terminate. (2) Else if $z_i = z$, then output z and terminate. (3) Else if $z_i = (r_1, r_2)$, then compute $\pi_i \leftarrow \mathcal{P}(\mathsf{crs}, (G_i, g_i'), \omega_i)$ and send $(\mathsf{claim}, sid, ssid, j, i, \phi_j, \tau, q, (\pi_i, r_j))$ to $\mathcal{F}^\star_{\mathrm{CR}}$, receive $(\mathsf{claim}, sid, ssid, j, i, \phi_j, \tau, \mathsf{coins}(q))$ and terminate.

Figure 3: Realizing DualEx with penalties.

DualEx with penalties: summary

- We set out to accomplish 2-party computation with
  1. security against active attacks.
  2. efficiency of semihonest protocols.
- We still make blackbox use of protocols that are secure against active attacks for the KT and SV invocations.
- The complexity of the functionalities KT and SV that ensure input/output consistency depends only on the input/output size of f, and not on the circuit complexity of f.
- For example, output size $= 1$ bit implies the least amount of secure computation in SV.
- Conclusion: if $|f| \gg |\text{input}| + |\text{output}|$, and both parties are honest, then the complexiity is essentially the same as that of the plain DualEx protocol.

Bitcoin Bounties

Pages: [1] 2 All — reply | watch | notify | mark unread | print

Author — Topic: REWARD offered for hash collisions for SHA1, SHA256, RIPEMD160 and other (Read 6263 times)

Peter Todd
Legendary
Activity: 896
aka retep
Ignore

REWARD offered for hash collisions for SHA1, SHA256, RIPEMD160 and other
September 13, 2013, 06:19:33 AM — quote #1

Rewards at the following P2SH addresses are available for anyone able to demonstrate collision attacks against a variety of cryptographic algorithms. You collect your bounty by demonstrating two messages that are not equal in value, yet result in the same digest when hashed. These messages are used in a scriptSig, which satisfies the scriptPubKey storing the bountied funds, allowing you to move them to a scriptPubKey (Bitcoin address) of your choice.

Further donations to the bounties are welcome, particularly for SHA1 - address 37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP - for which an attack on a single hash value is believed to be possible at an estimated cost of $2.77M (4)

SHA1:

```
$ btc decodescript 6e879169a77ca787
{
    "asm" : "OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP OP_SHA1 OP_EQUAL",
    "type" : "nonstandard",
    "p2sh" : "37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP"
}
```

⇒ 1) We advise mining the block in which you collect your bounty yourself; scriptSigs satisfying the above scriptPubKeys do not cryptographically sign the transaction's outputs. If the bounty value is sufficiently large other miners may find it profitable to reorganize the chain to kill your block and collect the reward themselves. This is particularly profitable for larger, centralized, mining pools.

Hash	37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP
Balance	2.47450702 BTC
Total received	2.47450702 BTC
Transactions	12

Bounty schemes

- The bounty maker M wishes to reward any bounty collector C upon producing a witness w that satisfies $\phi(w) = 1$.

Requirements of a noninteractive bounty protocol (informal)

- C can collect the reward even if M no longer exists.
- M cannot revoke the bounty before a witness w is found.
- M cannot deny payment to C once C reveals a correct w.
- Race-free: another collector C' cannot claim the reward after seeing the witness w that C claimed the reward with.

Private versus public bounties

- Private bounty: only M learns the witness w.
- Public bounty: everyone learns the witness w.

Why noninteractive bounty protocols are complicated

ZK contingent payments [Max11]

Two parties M, C_0 can run the following **interactive** ZK protocol:

1. C_0 proves in ZK to M that she knows w such that $\phi(w) = 1$ and $\texttt{AES}_k(w) = c$ and $\texttt{hash}(k) = h$, then sends (c, h) to M.
2. M creates a transaction that lets C_0 redeem coins if she signs with her secret key sk_{C_0} and provides x s.t. $\texttt{hash}(x) = h$.

The inherent problem with noninteractive bounties

- When the identity C_0 is known, we can require (asymmetric) signature of C_0 by hardcoding pk_{C_0} in the transaction script.
- When the identity C is unknown, a transaction whose only condition is $\phi(w) = 1$ can be hijacked once it is broadcasted and replaced with another transaction (with higher fee) that sends the coins elsewhere, before being buried under PoW.

Public bounty scheme

Public bounty protocol for the circuit $\phi(x, \cdot) = 1$

1. M creates secret keys sk, sk' such that $sk' = \text{puzz}(sk, t)$ can decrypt sk after t time by solving a *timelock puzzle*.
2. M uses *witness encryption* to create $\psi = \text{enc}_\phi(x, sk')$ that can be decrypted with a witness w that satisfies $\phi(x, w) = 1$.
3. M publishes the cipertext ψ, and broadcasts a transaction that can be redeemed by signing with sk and providing w such that $\phi(x, w) = 1$.
4. C computes w, then computes $sk' \leftarrow \text{dec}_\phi(\psi, w)$, then computes sk from sk' in t time, and redeems the transaction.

- If any collector C' sees the transaction that C broadcasted and tries to race for the reward, then C will have a head start of t time, and the PoW blocks that are solved during this time should make C's transaction irreversible.
- Realizable with current Bitcoin scripts? Depends on $\phi(x, \cdot)$.

Private bounty scheme

Private bounty protocol for the circuit $\phi(x, \cdot) = 1$

1. M creates a garbled circuit GC for $\phi(x, \cdot)$ such that I are the input labels of GC and e_0 is the output label of GC that corresponds to the value 1.
2. M creates a fresh secret key sk and uses *witness encryption* to create $\psi = \text{enc}_\phi(x, sk||I)$.
3. M publishes the cipertext ψ, and broadcasts a transaction that can be redeemed by signing with sk and supplying input labels that make GC produce the output label e_0.
4. C computes w, then computes $(sk, I) \leftarrow \text{dec}_\phi(\psi, w)$, then redeems the transaction by using sk and the input labels that correspond to w.
5. M reconstructs w by using I and the input labels that C supplied.

- Any other collector C' would still not know I, w, sk.

Thank you.

version 0.47 (without pauses)