The trust theory of money(TTM) – fix the money

Hadi Lashkari Ghouchani[1]

[1] *info@mail.trustycity.com*

2026-02-25

Modern Money Theory (MMT) is the dominant framework governing how we use money today. It is broad enough to partially incorporate the Credit Theory of Money (CTM) [1, 2, 3], which holds that money is fundamentally a form of credit and debt. However, we argue that MMT's half-baked implementation of CTM is causing serious problems for society, and that a full, digital implementation of CTM is needed. This sounds straightforward until we recognize that CTM is missing one crucial ingredient to be truly implementable—which is precisely why it remains only partially realized in MMT. CTM does not address how people can trust that a debt we call money will be repaid in the absence of a government guarantee. Searching for such an implementation leads us to the Trust Theory of Money (TTM), which also enables a novel digital realization. This approach is clearly distinct from blockchain-based solutions: cryptocurrencies discard the CTM entirely by grounding monetary value in scarcity, a position that directly contradicts the credit-and-debt foundation of CTM.

Keywords: **Modern money theory, Credit theory of money, Trust theory of money, cryptocurrency, blockchain**

1 Introduction

The anthropologist David Graeber asserted in his book[3] that the origin of the money was debt. We all read in the textbooks that the Persian Empire later introduced coins, marking the beginning of tying money to the state and the birth of fiat money. Since then we integrated the idea of fiat money into the modern money theory (MMT). Later Alfred Mitchell-Innes proposed the credit theory of money(CTM)[1, 2], arguing money is nothing but credit and debt, thus, the role of government is not necessary in the concept of money. However, CTM didn't suggest how to have a valuable money without a government!

Since the 2008 financial crisis, confidence in MMT has eroded significantly. In response, people anonymously implemented the blockchain – the underlying technology of Bitcoin and other cryptocurrencies. The core premise of blockchain is that no single entity can be trusted to issue money, so we avoid one by making digital scarcity. Most cryptocurrencies are accordingly described as zero-trust technologies. Their argument runs roughly as following: non-digital assets like gold are inherently scarce and appear to derive their value independently of any trusted authority such as a governments— then why not applying the scarcity as a principle to run the economy? The case of gold is very interesting. Gold cannot function effectively as money even though it is naturally scarce and therefore valuable. The reason is that a functioning economy requires the money supply to be adjustable by its issuer. That said, we need something "scarce enough", not absolutely scarce, to sustain a productive economy. Nevertheless, after more than a decade of experimenting with cryptocurrencies, an honest observer must concede that their users are still compelled to trust certain agents— those who have disproportionate computational power over ordinary users, or who are simply wealthy as measured in those cryptocurrencies. This implies that blockchain, in its best case scenario, secretly assumes that whoever holds computational power is a trustworthy economic actor, which is arguably a deeply flawed assumption.

Examining the concept of zero-trust more closely, we see that in MMT we still extend trust to institutions, assuming that governments and their bureaucratic systems are reliable, and that they can delegate this trust by issuing certificates to verified experts across various domains. Yet the very motivation behind the cryptocurrency movement is that these bureaucratic systems, as a trust maintainer in the society, have significant flaws.

The principal reason that bureaucratic systems fail to function reliably is captured by Goodhart's Law. Goodhart's Law [4] states that "when a measure becomes a target, it ceases to be a good measure." It claims measures should not be targets, they should rather be checkpoints. But for an individual, a certifi-

cate of accomplishment is precisely a target, and so people find ways to obtain these credentials without necessarily possessing the skills they are supposed to certify. It is also worth noting that in practice there is no comparable measure of trustworthiness applied to governments, even though individuals are regularly tested and verified. The current system still operates on trust—it is simply not a very reliable form of trust.

A problem that has yet to be acknowledged in monetary theory—at least as far as the author is aware—is this: selling any good or service in an economy requires some degree of trust between buyers and sellers, as well as among producers and their supply chains. People clearly express this trust through brand loyalty, which is the central function of the concept of a brand, yet in theory we clearly do not formally account for this trust when calculating the exchanged value in a deal.

It is also worth noting that CTM and TTM (discussed in section 2) are already partially incorporated into MMT. Credit cards and credit scores, for instance, are designed around the idea of debt as money, built on top of a system of mutual trust among all parties in an economy. Yet fiat money—the value stored on a debit card—is treated as categorically separate, as though we trust governments unconditionally and need not track any government "credit score."

A final concern is that in the past decade, particularly since 2020, the economy has been subject to a succession of hype cycles. Those who ride these waves of hype often end up capturing the most capital in society, which they then deploy to generate further hype rather than to produce genuinely valuable goods or services. In theory, the market should self-correct when these inflated promises fail to materialize. Instead, the architects of these cycles take ordinary people's jobs and livelihoods hostage, forcing governments to absorb their failures. Left unchecked, accumulation of their failures will only end up with a failure of the whole society.

2 Trust theory of money

This is where the Trust Theory of Money (TTM) enters the picture. ***TTM is a patch to CTM: it asserts that to assign value to a debt, we must trust the entity that issued it.*** We trust governments—even in non-democratic countries, where that trust is enforced by military power— to assign value to the fiat money they issue. We also trust them to understand how money works to avoid printing unlimited quantities and thereby destroying its value. The principle of scarcity emerges in TTM not from an inherent physical property, but from the trust we place in the issuer's understanding of how to keep their debt valuable.

Under TTM, we claim ***that trust is absolutely necessary for a productive economy***, which means that even in the current system we have already implemented some form of trust system. Bureaucratic certificates are the unit of measure for trust in the current system.

Beyond certificates, we also have credit scores to measure how accountable a person is for their debts—in other words, how trustworthy they are when it comes to repaying what they have borrowed. The noteworthy feature of the credit score is that it is not a target: it is continuously updated based on an individual's behavior over time, embedded in the natural flow of life. The data feeding the credit score is gathered at checkpoints, which makes it resistant to Goodhart's Law [4]. This is the key insight. We should move toward using more checkpoint-based data to evaluate people's trustworthiness.

2.1 Digital TTM — its requirements

Here we develop a digital implementation of TTM. The first step is to enumerate the requirements for this implementation. We want the behavior of this digital implementation to resemble the experience of a person walking into a store and paying in cash. The customer does not conceal their identity, and the store does not hide its sign at the moment of the transaction, but to an outside observer it is not clear what was purchased or how much was paid. We call this the *customer-store-cash scenario*, and more broadly the *privacy principle*, which must be satisfied by our algorithm.

We deliberately avoid radical departures from the current system to distinguish this solution from blockchain technologies, because its architecture differs too fundamentally from established practices. Since TTM doesn't require scarcity as a principle, there's not need for a *central database* we call blockchain. Another reason to walk away from blockchain is that in most cryptocurrencies all transactions on a blockchain are stored in plaintext, which clearly violates users' privacy above.

In the customer-store-cash scenario, the store, and later other parties like government, can verify the customer actually paid by just looking at the coins or bills in the store owner's hand. The type of algorithm we are seeking here is a non-interactive Zero-knowledge proofs (ZKPs)s[5], where nobody can look inside the wallet of customer, or the store, to see their balance, but later they can verify the customer had non-negative balance, since the transaction happened and store can use the received cash to buy from others.

It's important to mention there are cryptocurrencies already incorporated the idea of non-interactive Zero-knowledge proofs to hide the balance or other information, however, these cryptocurrencies are still working on top of blockchain and scarcity principle, so they are relying on the secret assumption that mentioned before: holders of the blockchain in a network are trustworthy, where they would not admit it since

they have been promised a zero-trust solution, where TTM claims it's impossible!

For instance, zk-SNARKs algorithm[6], which is also a non-interactive Zero-knowledge proof algorithm used in Zcash. Later we will mention that TTM's algorithm has better performance respect to zk-SNARKs, but zk-SNARKs can be used for more diverse messaging, where TTM's algorithm only hides the balance of a wallet.

The next requirement help us understand why we avoid calling the TTM's implementation below a cryptocurrency, This implementation should behave like fiat money as closely as possible. Fiat money functions like a spendable note issued by the government: we trust the government to accept it as payment for goods, services, or taxes. By "spendable" we mean that you can transact using a fraction of a note— you simply make change—but the crucial point is that the value is fungible and immediately usable. This is true of government-issued fiat money, however, in the CTM what government have been issuing was a form of debt, which is more like issued cheques, but currently, cheques issued by private individuals are not spendable. An individual's cheque is not spendable in this sense: you must wait until the date specified on the cheque to cash it, and in the meantime you cannot use any portion of it to purchase goods or services.

This requirement under the umbrella of CMT unifies the concept of currency and cheque, where we use this to distinguish this solution from cryptocurrencies, by calling it a *cryptocheque*.

In TTM, the issuer of the spendable cheque is an individual trusted by both parties of a transaction. When the recipient presents the cheque to the issuer at the specified time, the issuer is pledged to redeem it and settle the underlying debt.

Let us work through a detailed scenario. First, there is an issuer—call them Γ— with a publicly available track record of honoring their debts. Γ issues a large cheque together with a pledge. We refer to the unit of value this cheque represents as κ, which you can think of as a currency name. In the pledge, Γ promises that whoever returns κx (i.e. x units of this cheque) after a specified time will receive y units of something of value. The cheque is therefore said to be backed by something tangible. The specified time, along with any other conditions, can be legally formalized and backed by the government's legal system. However, if Γ fails to honor the agreement, the system records this failure, and people will subsequently extend less trust to Γ— very much like the current credit score system.

Γ spends the cheque to acquire goods and services from people who have reviewed their track record and trust Γ sufficiently. One such person is Alice (A), who now holds κa. Alice wants to spend κt (where $\kappa t \leq \kappa a$) to purchase something from Bob (B), who also trusts Γ sufficiently. If B does not trust κ, then A must use a different currency by exchanging cheques or using some other method. If A and B share no common cheque whose issuer they both trust sufficiently, they simply cannot transact.

The requirements for this system are as follows:

- All parties—Γ, A, and B— are publicly identified. No anonymity. (Recall the customer-store-cash scenario above.)
- Only A and B should know the amount of the transaction, d_{AB}, as required by the privacy principle. The other parties, including Γ, should not be able to calculate d_{AB}.
- B should be able to independently verify that A has sufficient funds to complete the transaction, without looking inside A's wallet.
- Additionally, B should be able to independently verify consistency of the initial and final states of A's and B's wallets Other parties, including Γ should be able to verify B's calculations.
- A should be able to spend a fraction of the value in her wallet.
- Γ is a sufficiently trusted entity, so it is acceptable to query its public API for the latest verified state of A's wallet, but as mentioned Γ should not be able to look inside the wallet. Furthermore, we must be able to track Γ's record to evaluate its trustworthiness.

2.2 TTM algorithm

The next step is to fill in the technical details: what data is needed, which data is public, and which is private. To implement the privacy principle, we require an algebraic structure with a homomorphism [7] with respect to addition. We use modular arithmetic, very similar to RSA [8], to enforce the privacy principle as described in the customer-store-cash scenario. Ultimately, the main goal of this paper is to demonstrate that an implementation satisfying the requirements above exists.

The process begins as follows. Γ drafts and legally signs the pledge agreement. It must hold a value, for instance by specifying the date on which the debt will be repaid, the total value of the cheque, and the redemption amount per unit of κ issued. Γ then generates two large random numbers, c_p and p_p (with $c_p \ll p_p/2$), where c_p is the base for all computations and p_p is a prime number used as the modulus. Based on the pledge agreement, Γ commits to a total value γ_p by encrypting it.

$$w_p \equiv c_p^{\gamma_p} \pmod{p_p} \tag{1}$$

Here w_p is the encrypted wallet. In general, wallets are encrypted as $w_p \equiv c_p^x \pmod{p_p}$, where x is the private credit value—invisible to the public even when w_p is published. Note that the subscript $_p$ denotes public data; similarly, $_A$, $_B$, and $_{AB}$ denote data that is private to A, B, or both, respectively.

Since the cheque is spendable, Γ can spend it much like a credit card in the current system. We now de-

scribe how a transaction between A and B takes place. Note that Γ itself can be either A or B, so the following also covers how Γ would spend the cheque.

Returning to the customer-store-cash scenario, with Alice as the customer and Bob as the store: suppose Alice has selected an item and is ready to pay using κ, which both A and B trust sufficiently. Alice scans Bob's address (and any public keys used by other encryption algorithm) to initiate the payment. At this point Alice has Bob's address, the amount she wishes to transfer, and her own wallet $wa1_p \equiv c_p^{a1_A} \pmod{p_p}$. She then computes the following.

$$t_p \equiv c_p^{(d_{AB})} \pmod{p_p} \quad (2)$$
$$dn_{AB} \equiv -d_{AB} \pmod{p_p - 1} \quad (3)$$
$$ti_p \equiv c_p^{(dn_{AB})} \pmod{p_p} \quad (4)$$
$$wa2_p \equiv wa1_p \times ti_p \pmod{p_p} \quad (5)$$

Here d_{AB} is the transaction amount to be moved from Alice's wallet, $wa1_p$, to Bob's wallet, $wb1_p$. After the transaction is sealed, Alice's wallet becomes $wa2_p$ and Bob's becomes $wb2_p$. Importantly, d_{AB} is visible only to Alice and Bob, satisfying the privacy requirement.

The term t_p is the encrypted transaction amount and is visible to everyone. Alice will send all of the public variables she has computed (those subscripted $_p$) to Bob so that he can verify the validity of the transaction. Before doing so, however, she must run the extended Euclidean algorithm to find the greatest common divisor (GCD) of the previous value in her wallet and the new value after the transaction.

$$s1_p, s2_p, s3_A = extended\_euler\_gcd(a1_A, a1_A - d_{AB}) \quad (6)$$

This must satisfy the Bézout's identity, $s1_p \times a1_A + s2_p \times (a1_A - d_{AB}) = \gcd(a1_A, a1_A - d_{AB}) = s3_A$. This step is the computational bottleneck of the algorithm, with $O(\log(\min(a1_A, a1_A - d_{AB})))$ time complexity, which is entirely practical. Using these secret values, Alice can then compute the following public helper variables.

$$h1_p \equiv \begin{cases} s1_p \pmod{p_p - 1} & if s1_p \geq 0, \\ -s1_p \pmod{p_p - 1} & if s1_p < 0. \end{cases} \quad (7)$$

Then similarly

$$h2_p \equiv \begin{cases} s2_p \pmod{p_p - 1} & if s1_p \geq 0, \\ -s2_p \pmod{p_p - 1} & if s1_p < 0. \end{cases} \quad (8)$$

The following value, however, is kept private:

$$s4_A \equiv \begin{cases} s3_A \pmod{p_p - 1} & if s1_p \geq 0, \\ -s3_A \pmod{p_p - 1} & if s1_p < 0. \end{cases} \quad (9)$$

Using $s4_A$, we compute the final public helper variable:

$$h3_p \equiv c_p^{(s4_A)} \pmod{p_p} \quad (10)$$

Notice, $s1_p$, and $s2_p$, are public, but $s3_A$ and $s4_A$ are only visible to Alice.

After computing all of the above, Alice signs them and transmits them to Bob for verification. In the next step, Bob queries Γ's public API for the latest state of Alice's wallet to verify that Alice was truthful about $wa1_p$. Bob can independently recompute several of these values before processing Alice's data:

$$t_p \equiv c_p^{(d_{AB})} \pmod{p_p} \quad (11)$$
$$dn_{AB} \equiv -d_{AB} \pmod{p_p - 1} \quad (12)$$
$$ti_p \equiv c_p^{(dn_{AB})} \pmod{p_p} \quad (13)$$
$$wa2_p \equiv wa1_p \times ti_p \pmod{p_p} \quad (14)$$
$$wb2_p \equiv wb1_p \times t_p \pmod{p_p} \quad (15)$$

Bob must also verify that Alice had sufficient credit in her wallet, by checking the following conditions:

$$0 < h1_p < p_p/2 \quad (16)$$
$$p_p/2 < h2_p < p_p \quad (17)$$
$$wa1_p^{(h1_p)} \times wa2_p^{(h2_p)} \equiv h3_p \pmod{p_p} \quad (18)$$

Since all of these parameters are public, any third party can independently verify the transaction without ever learning the value of d_{AB}. By checking the conditions above, we confirm that Alice's wallet did not go negative after the transaction.

After verification, Bob signs the data and posts it to Γ's API to update both wallets. Γ can also run all the verifications independently; in the event of a failure, Γ has full authority to revert the transaction, since it was Bob who failed to verify correctly and Bob who loses the credit. On the other hand, Bob has the non-interactive Zero-knowledge proofs (ZKPs)s to show everyone in case of Γ's misbehavior to have impact on Γ's trustworthiness.

Γ's API should permit independent parties to run the above verification on each transaction, in order to maintain Γ's trustworthiness.

Note that Γ's API may lock Alice's wallet for the duration of a transaction (with a sensible timeout). Since Γ receives one request from Bob to fetch the latest state of Alice's wallet, and a subsequent request once verification is complete, the time interval of the transaction is transparent to Γ.

To improve security, larger numbers are preferable. We can achieve this by bit-shifting γ_p and all d_{AB} values to the left by a constant ls_p defined in the pledge agreement, then adding a small random salt to each d_{AB} with an upper bound of m_p. As a result, a given cheque can support at most $2^{ls_p}/m_p$ transactions before the accumulated salts encroach on the actual

credit value in the wallets. The trade-off between a bounded transaction count and significantly stronger security is well justified.

2.3 Attacks

The main attack vectors are as follows.

2.3.1 Γ proves less trustworthy than A and B assumed.

This can be mitigated by gradually increasing transaction limits as track records accumulate, and by incorporating credit scores. Furthermore, if Γ wishes to increase their trustworthiness, they must share more information. This creates a natural balance in the system: all transactions are protected, but parties who share more data gain more trust from others. More broadly, this balance is related to how any productive system—including the human brain— operates at a boundary that is significant for its fractal properties. One such property of a fractal is the well-defined maximum reach, which aligns naturally with the principles of a free market.

2.3.2 Cyberattacks on Γ's API (e.g., DDoS)

Established API best practices can defend against the majority of such attacks.

2.3.3 Brute-force attacks

Brute-force attacks become feasible only if c_p, p_p, or m_p are small. For example, if m_p is a 1024-bit number and we allow up to 2^{1024} accumulated transactions per cheque, then $ls_p = 2048$ (a 2048-bit left shift), we can conclude that γ_p should be less than a 2047-bit number ($4096 - 1 - 2048 = 2047$) for a practical and secure implementation. In a single core of a $5.0GHz$ CPU, Alice's and Bob's computations altogether takes less than a seconds.

This is also where we can compare the performance of TTM's algorithm and zk-SNARKs. With above assumptions about c_p, p_p, and m_p, you can say it's proof size is about 6×4096 bits, or 24 bytes, where the proof size of zk-SNARKs is around 200-300 bytes.

2.3.4 Recursive attack

In the Bézout's identity, $s1_p \times a1_A + s2_p \times (a1_A - d_{AB}) = \gcd(a1_A, a1_A - d_{AB}) = s3_A$, we can hide $s3_A$, but $s1_p$ and $s2_p$ are publicly computable. This means some one can use the 'extended_euler_gcd' function for $s1_p$ and $s2_p$ to find their GCD along with coefficients of $a1_A$ and $a1_A - d_{AB}$, which supposed to be only visible to A. Additionally, if $\gcd(a1_A, a1_A - d_{AB}) = 1$, which means $a1_A$ and $a1_A - d_{AB}$ are coprimes, then both of them will be revealed. To avoid $\gcd(a1_A, a1_A - d_{AB}) = 1$ Alice can simply regenerate the salt to make sure GCD is not one. This will mitigate the first problem.

However, you can see each transaction chains $(a1_A, a2_A)$ pairs— $a2_A = a1_A - d_{AB}$— where $a1_A$ and $a2_A$ are the balances of the Alice's wallet before and after a transaction. If one of them in the chain become revealed, then attacker can exploit it to compute all of them, but in fact, one of them is public by design, which is γ_p— the initial amount of the cheque in the pledge agreement. To mitigate this problem, Alice, as the issuer whose balance is public, can hide $h1_p$, so it would be $h1_A$ now, then share its encrypted version $c_p^{(h1_A)}$, then Bob could use this to verify the transaction. In fact, it makes sense for the issuer to split its wallet randomly into two using this method just after issuing the cheque— to encrypt all the transactions afterward.

2.3.5 Quantum attack, and specifically the Shore algorithm[9].

This construction is vulnerable to quantum attacks on RSA: given that the recursive attack above is depending on the factorization of big numbers.

3 Discussion

The author is currently developing a software infrastructure to implement this solution, with particular focus on the checkpoint data-gathering component so that users can evaluate one another with greater confidence. This infrastructure will include B2B components— such as libraries and APIs for banks and credit bureaus— enabling faster system integration. It will also include B2C components, such as mobile applications, allowing users to analyze and interpret trust data more reliably.

The author hereby claims all rights to the implementation of this algorithm, to enable open publication without forfeiting credit. It is worth noting that the person who builds the infrastructure for your trust system should themselves be trustworthy. The author's long-term aspiration is a society in which ideas can be published freely and openly without the risk of losing attribution—which is itself one of the ultimate goals of implementing TTM.

4 Conclusion

Although cryptocurrencies consistently claim to decentralize money, TTM in fact supports an arbitrary number of independent currencies and cheques, making it far more decentralized than any cryptocurrency has achieved. Decentralization matters because it is the foundation of a genuinely free market. The core idea is that money is a form of credit and debt, established through trust among the parties to a transaction in a

productive economy. We have presented an algorithm that unlocks the potential of the digital world to operate as money was originally intended.

Bibliography

[1] Alfred Mitchell-Innes. "What is Money?" In: (1913). URL: https://www.newmoneyhub.com/www/money/mitchell-innes/what-is-money.html.

[2] L. Randall Wray. *Credit and State Theories of Money: The Contributions of A. Mitchell Innes*. 2004. ISBN: 9781843765134.

[3] David Graeber. *Debt: The First 5,000 Years*. 2011. ISBN: 9781933633862.

[4] Charles Goodhart. "Problems of Monetary Management: The UK Experience". In: (1975). URL: https://www.econbiz.de/Record/problems-of-monetary-management-the-u-k-experience-goodhart-charles/10002525062.

[5] Oded Goldreich. *Foundations of Cryptography Volume I*. 2001. ISBN: 978-0-511-54689-1. URL: https://www.cambridge.org/core/books/foundations-of-cryptography/B61B6AD235D2034D511A5FF740415166.

[6] Jens Groth. "On the Size of Pairing-based Non-interactive Arguments". In: (2016). URL: https://eprint.iacr.org/2016/260.pdf.

[7] Robert Fricke. *Vorlesungen über die Theorie der automorphen Functionen*. 1897. URL: https://archive.org/details/vorlesungenber01fricuoft/page/n5/mode/2up.

[8] Leonard Adleman Ron Rivest Adi Shamir. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: (1977). URL: https://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TM-082.pdf.

[9] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: (1997).

A Supplementary Material

Before starting of the transaction both Alice and Bob know their wallets.

```
def before_transaction(
  c_p, # base
  p_p, # modulo
  a1_A, # the credit in Alice's wallet before transaction
  b1_B # the credit in Bob's wallet before transaction
):
    wa1_p = mod_pow(c_p, a1_A, p_p)
    wb1_p = mod_pow(c_p, b1_B, p_p)
    return wa1_p, wb1_p
```

When transaction starts, Alice computes the following.

```
def alice_transaction(
  c_p, # base
  p_p, # modulo
  a1_A, # the credit in Alice's wallet before transaction
  wa1_p, # the Alice's wallet before transaction
  diff_AB # the transaction's value
):
    neg_diff_AB = mod_neg(diff_AB, p_p - 1)
    t_p = mod_pow(c_p, diff_AB, p_p)
    ti_p = mod_pow(c_p, neg_diff_AB, p_p)
    wa2_p = mod_mul(wa1_p, ti_p, p_p)
    s1_p, s2_p, s3_A =
    ↪ extended_euler_gcd(a1_A, a1_A -
    ↪ diff_AB)
    h1_p = (s1_p if s1_p > 0 else -s1_p) %
    ↪ (p_p - 1)
    h2_p = (s2_p if s1_p > 0 else -s2_p) %
    ↪ (p_p - 1)
    s4_A = (s3_A if s1_p > 0 else -s3_A) %
    ↪ (p_p - 1)
    h3_p = mod_pow(c_p, s4_A, p_p)

    return t_p, ti_p, wa2_p, h1_p, h2_p,
    ↪ h3_p
```

Then transmit these data to Bob, where Bob verify them as following.

```
def bob_verification(
  c_p, # base
  p_p, # modulo
  wa1_p, # the Alice's wallet before transaction
  wb1_p, # the Alice's wallet before transaction
  diff_AB, # the transaction's value
  at_p, # encrypted transaction computed by Alice
  ati_p, # encrypted transaction computed by Alice
  awa2_p, # the Alice's wallet after transaction computed by Alice
  h1_p, # helper variable one
  h2_p, # helper variable two
  h3_p # helper variable three
):
    assert diff_AB != 0
    neg_diff_AB = mod_neg(diff_AB, p_p - 1)
    t_p = mod_pow(c_p, diff_AB, p_p)
    assert t_p == at_p
    ti_p = mod_pow(c_p, neg_diff_AB, p_p)
    assert ti_p == ati_p
    wa2_p = mod_mul(wa1_p, ti_p, p_p)
```

```
    assert wa2_p == awa2_p
    assert 0 < h1_p < p_p // 2
    assert p_p // 2 < h2_p < p_p
    assert 0 == mod_sub(
        mod_pow(wa1_p, h1_p, p_p) *
        ↪  mod_pow(wa2_p, h2_p, p_p),
        h3_p,
        p_p
    )

    wb2_p = mod_mul(wb1_p, t_p, p_p)
    return wb2_p
```