1 Bolt Overview

This section is an overview of the scheme that is presented in the original Bolt paper so that we are all on the same page. The paper actually outlines a few different protocols with optional configurations so this only describes the configuration that is relevant to us.

1.1 Micropayment Primer

: There are three roles in a micropayment network.

- End Users: These users send payments to each other by updating payment channels with mutual intermediaries.
- Intermediary: These are high-bandwidth nodes who route payments between end users.
- Ledger: This party maintains the global state of the network. It resolves conflicts betwen end users and intermediaries by executing an agreed-upon resolution procedure.

  In our particular application, our end users will primarily be clients and relays. We will use C, R, and I to denote clients, relays, and intermediaries, respectively.

Every payment in a micropayment channel network needs to route payment information in two separate channels, one between C and I and one between I and R. In such a network, the challenge is to ensure that both legs of the payment complete atomically such that no party can cheat by prematurely aborting. In an anonymous micropayment network, the additional challenge is to make each payment unlinkable with any past or future payments.

1.2 Procedure:

1. At the start of time, the ledger either runs a trusted setup to generate public parameters for the system. These are used to form zero-knowledge proofs throughout.
2. I runs an $Init$ procedure then escrows some amount of funds to the ledger. The amount that I is able to escrow effectively determines the throughput of payments that it can handle. Since there is an opportunity cost to locking up money, the intermediary should charge some fee for its services.
3. End end users (C and R) similarly run their own $Init$ procedure to escrow some amount of money to the ledger.

4. Each user runs an *Establish* protocol along with I to turn their escrowed funds into a micropayment channel. The user receives a "wallet" which is a piece of data that encodes their balance on the channel. At this phase, the balance of the wallet is simply the amount that the user originally escrowed.

5. Suppose that C would like to pay some relay R that is also connected to I, then the three of them initiate a *Pay* protocol. C creates a zero-knowledge proof that he owns a valid wallet (with a balance B_C) without revealing the actual balance. He then sends it to I along with a commitment to a new wallet that is updated with the new balance $(B_C - \epsilon_C)$. R does the same thing. I then sends C a "refund token" which gives C the power to close the channel and collect exactly $B_C - \epsilon_C$ From the ledger. Keep in mind that I should only do this because C the one paying I. I cannot simply send a similar refund token to R, otherwise R might collect the payment while C aborts and keeps his old wallet instead. To protect its money, I sends R a conditional refund token which says that R can only collect funds if it also reveals proof that C has revoked his old wallet. To continue the protocol, C then sends a "revocation token" to R. R now has the power to close out her channel and collect $B_R + \epsilon_R$ funds.

   Note that ϵ_C and ϵ_R do not have to be equal. If ϵ_C is higher in value, then I gets to keep the remainder of the transaction as a fee.

   While all parties now have the necessary tokens to cash our their microchannels, they will typically opt to keep it open. The intermediary, having obtained proof of revocation of C's old wallet, can safely issue C a new one with the updated balance by way of a blind signature. Similarly, I also provides R with an updated wallet. These new wallets are completely unlinkable to the old wallets at the beginning of the interaction and in this way, C and R can continue making anonymous payments so long as can prove ownership of sufficient funds on their channel.

6. In the event of a dispute or intentional channel closure, an end user can post their revocation token along with other data to submit a claim on the ledger. If this is an invalid (perhaps outdated) token, then I should be able to refute the claim using the revocation token that it should have saved. The ledger verifies the validity of these tokens and makes a fair decision on the final payouts.

Security:: Ledgers are trustless in the sense that everything they do is publically verifiable. Intermediaries are trustless in the sense that they are self-interested. Assuming the end users also follow the protocol, I cannot steal money. Any deviation on the part of I will only cause them to lose money. A payment by an end user is anonymous because neither the sender nor recipient can be identified from the pool of end users that are connected to I.

2 Nanopayment Overview

This section describes a nanopayment extension to Bolt which allows for linkable but extremely fast payments between end users in a micropayment network.

2.1 Mo Anonymity, Mo Problems:

It is debatable as to how "micro" Bolt micropayments actually are. Since they did not provide an implementation, there are no hard numbers. However, if we trust them on their knowledge of the cryptographic primitives, then they make some qualitative remarks that we can reinterpret. Here is by slightly-informed guess of the performance of a single execution of the *Pay* protocol:

- Number of messages: ≈7
- Bandwidth: ≈1 kilobyte
- Client Computational Time: ≈1 second

These costs are arguably a bit steep. It's also unlikely that we would be able to make any direct improvements on their protocol since the authors really seem to know what they are talking about.

2.2 Solution

The good news is that Bolt has more anonymity then we actually need. We can exploit this fact to improve efficiency for certain types of transactions. In Bolt, every single payment is totally unlinkable to any prior payment to anyone but the payer. That would not strictly be necessary for Tor payments. A relay that supports a circuit definitely know that any messages coming along that circuit are coming from the same (albeit anonymous) client. Rather than routing a stream of expensive micropayments through a circuit, we can instead set up a transparent "nanoypayment" channel. This channel can be used to make very fast incremental payments between the same client and relay. It doesn't matter that these intra-circuit payments are linkable as long as the original channel setup and closure procedures remain anonymous.

The process to set up, use, and close down a nanopayment channel are meant to replace an instance of the Bolt *Pay* protocol. Assume that all parties already successfully created micropayment channels using the standard of the Bolt *Init* and *Establish* procedures. With a given wallet, a user can *either* make a micropayment *or* set up of a nanopayment channel. The user alternate make micropayments and setting up nanopayment channels in any order as long as they still haves on their micropayment balance. Nanopayment channels, therefore, are just a specialized type of micropayment.

2.3 Procedure

1. At this point, C should have an open micropayment channel with the intermediary and therefore access to a valid wallet. C first computes a hashchain of some desired length based on the number of nanopayments he would like to be able to make. C then uses this hash chain along with his current wallet to initiate a $nInit$ protocol with the intermediary. Just like in the Bolt Pay protocol, the client receives a refund token that allows him to claim funds on the ledger. This token allows him to retrieve some value between B_C and $B_C + \epsilon_C$. The variable n is the number of hash preimages C is willing to reveal if he ends up posting the refund token to the ledger.

2. Through the $nInit$ protocol, C has the ability to actually set up a nanopayment channel with any other end user at some point in the future. As soon as C wishes to do this, he runs the $nEstablish$ protocol with, for instance, a relay R. In this protocol, C sends some information about his wallet and the hash chain to R. C also sends a revocation token invalidating his old wallet. He can do this because he already possesses I's refund token for the payment channel.

   When R receives this information, she has no way to confirm its validity. She must forward it to I so that I can check that it has some record of this nanopayment channel. If I is satisfied with authentication of the channel, then I confirms with R via a signed message. Also included in the message is a conditional refund token for R's channel. The token reads the following:

   R is entitled to collect $B_R + \delta_R * k$ funds from the channel so long as she provides:

   - A revocation token on C's old wallet
   - The k^{th} preimage in the nanopayment hash chain

3. At this point, R is convinced of the validity of the nanopayment channel. To make a payment of value δ, C simply pops off the next preimage of the hash chain and sends it to R. R verifies the hash and accepts the payment. This can be repeated any number of times (up to n) to make each incremental payment in the channel

4. At any time, R has all of the information necessary to cash out on the ledger and claim the most recent amount. If she does not wish to do this, then she can instead keep the channel alive by initializing a $nClose$ protocol with I [[FR: This is ambiguous: you keep the micropayment channel alive but close the nanopayment channel, right?]] [[TD: Yes! Sorry, I changed the previous sentence to say "$nClose$", which closes out the nanopayment channel but keeps the micropayment channel]]. R simply sends I information about the latest preimage that she received from

C. I needs this information so that it can refute any false claims by C about how many preimages he did or did not send to R. Now that both parties have the most updated information, they engage in a standard Bolt Pay protocol to issue a new anonymous wallet to R containing a balance $B_R + \epsilon_R$ where ϵ_R is simply the amount that was transferred on the nanopayment channel ($\delta_R * k$). I is happy because all prior wallets with R have been revoked and R is happy because she now has a fresh new anonymous wallet to use.

[[FR: We don't want relay's funds to be anonymous - any transaction to a relay should be observable (including the amount). Could we cash-out to a public wallet?]] [[TD: Yeah, that should still work. In that case, only the sum total that they made over the microchannel will be revealed.]]

[[FR: What's the cost of the R-side in the Pay protocol?]] [[TD: Basically negligible :) Since everything was frontloaded in the $nEstablish$ phase, the R-side payment is literally just the following:
* $if(Hash(new\_payment) == last\_payment)$
* $\quad last\_payment = new\_payment$
Maybe we might want to send a confirmation message back to C or something at some point, but otherwise that's it.]]

[[FR: How do we enable the Tax system with this? The Pay protocol could involve a third-party (tax collector)?]] [[TD: The tax would probably be collected when an intermediary and an end user complete a micropayment $Close$ protocol, since that's the first time that anyone will be able to use their money anyways. We could force the intermediary to close out to a transparent account and have the ledger implement a rule so that some percentage of intermediary closures automatically get awarded to a Tor authority account.]]

5. Similarly, C can complete a similar $nClose$ protocol with I. However, it is important to I that C closes his channel after R closes hers. Just as before, I leaves this protocol with the assurance that all of C's prior wallets have been revoked and C leaves with a fresh wallet containing $B_C + \delta_C * k$ funds. Since the order of the closure protocols is important, R can enact a petty denial of service attack on C by refusing to close her channel. We will have to combat this by adding a "timeout" condition by which R forfeits the nanopayments she received if she does not close within a reasonable time.

6. In the event of channel closure, the procedure is similar to the one outlined in Bolt. The end user posts some information containing the refund token that they should have obtained from I and I gets a chance to refute the claim if necessary. When C wishes to close a nanopayment channel on the ledger, then he posts his refund token along with the last preimage that *he claims to have sent to* R. If I agrees with the number of pay, then no action is necessary. However, if I disagrees, then it should have some

record of an earlier preimage it received from R. This would allow I to collect its fair share along with some penalty on C's attempt to cheat.

2.4 Security

The econonmic security of the nanopayment extension is identical to Bolt. The anonymity appears to be identical as well with the exception that I can now observe the number of payments made in each nanopayment channel. Assuming nanopayment channel lifetimes map exactly to Tor circuit lifetimes, I effectively learns the following piece of information that it otherwise would not have in the current version of Tor:

Somewhere among my pool of end users, there exists a circuit X that lasted for k payments across t time for v value.

Some notes about the above statement:

- It would be possible to add some uncertainty to I's knowledge about circuits by initiating and closing multiple nanopayment channels in a single circuit or maintaining the same nanochannel across multiple circuits (for instance with long-lived entry guard relationships). However, this may not be optimal from a performance standpoint.
- In Bolt, the monetary value transferred in a micropayment can be hidden from I. It's *probably* possible to adapt this feature for nanopayments as well so that I in fact does *not* see the value v in the above statement.

The information learned by I is no more than what any single relay on the circuit already knows. Thus, it might be useful to model I as a 4^{th} relay on the circuit from an anonymity standpoint.

[[FR: I disagree on the Tor-related security claim. What I learns alone is no more than what a single relay learns, as you said. However, what colluding Is learn is the secrecy of your circuit, assuming that $Intermadiaries$ correlate perfectly and instantaneously, which is a widely used threat model assumption in the Tor community. As you said in your preamble, $Intermediaries$ would be high-bandwidth nodes. Consequently, the secrecy of Tor circuits would be backed by a few high-bandwidth nodes?]] [[TD: Hmm... well what if we ammend the statement to be "I leans no more than what any *middle* relay already knows"? As far as I can tell, no amount of colluding middle relays can deanonymize anybody. Same with the intermediary. Even if we have a single intermediary (the ultimate "colluding intermediaries" scenario), they would just see a bunch of circuits popping into existence without know who they are associated with. Would that be problematic? I would all of a sudden have access to some cool statistical information about Tor usage, but it feels like that is okay as long as the privacy of the individual client get preserved, which it is.]]

[[FR: Why would we use an Intermediary to pay the Guard relays since it already

knows us? Could we directly open a nanopayment channel to our Guard, giving it a wallet? If it cheats and we detect a problem with the balance when we close the nanopayment channel, we just rotate the Guard relay and fire a complain to the dirauths (that could help to exclude it from the network). When we have non-cheater guard relay, we keep it lifetime as this is anyway what Tor is moving to. The objective here is simply to reduce CPU cost when we don't really need what Bolt offers.]] [[TD: Yeah that's a great point, actually. The entire point of the intermediary business is that a longer relationship with the intermediary allows us to limit the size of the ledger, but if we have an equally long relationship with the Guard then it makes total sense to skip the overhead. I'm on board with this change!]]

[[FR: To fix both above problems, I would suggest to do the following: direct payment to the Guard and i_{th} relay of the circuit used as Intermediary of the $i+1_{th}$. Exit relays would not have to handle any Intermediary job (they already have too much pressure) and this would give the same circuit secrecy as Tor has today. However, the CPU cost of intermediaries is moved to Guard and Middle relays and Tor clients need to run the establish channel procotol with each of them at least once in days/weeks/months. There is some trade-off there between cumulative deposit and CPU cost for relays, if such thing is doable in practice...]] [[TD: I'm not sure I understand... could you elaborate? It vaguely sounds like we would be switching intermediaries every circuit, which would be very costly for the ledger]]

3 API

Notation remarks: Suffixes and prefixes of C, R, and I correspond to variables relating to the client, the relay, and the intermediary, respectively. Both clients and relays are considered to also be "End Users", denoted E. Function definitions prefaced with $(\forall E)$ should be understood to mean that the procedure is meant to be run once for both the client and the relay separately (i.e. $\forall E \in \{C, R\}$). Finally, the notation $I{:}E$ marks a variable that "belongs" to I but is associated with a channel that it maintains with E.

3.1 Bolt Procedures

$\text{KeyGen}(pp) \rightarrow (pk, sk)$

$\forall E$: $\text{Init\_E}(pp, B_0^E, B_0^{I:E}, pk_E, sk_E) \rightarrow (T_E, csk_E)$

$\forall E$: $\text{Init\_I}(pp, B_0^{I:E}, B_0^E, pk_I, sk_I) \rightarrow (T_{I:E}, csk_{I:E})$

$\forall E$: *Protocol*: Establish

$\text{End\_User}(pp, T_{I:E}, csk_E) \rightarrow (pp, w_E)$

$\text{Intermediary}(pp, T_E, csk_{I:E}) \rightarrow (established)$

Protocol: Pay

Client$(pp, \epsilon_C, w_C) \rightarrow (w_C, rt_C)$

Relay$(pp, \epsilon_R, w_R) \rightarrow (w_R, rt_R)$

Intermediary$(pp, Commit(\epsilon_C), Commit(\epsilon_R), S_I) \rightarrow (S_I)$

$\forall E$: Refund$(pp, T_{I:E}, csk_E, rt_E) \rightarrow (rc_E)$

$\forall E$: Refute$(pp, T_E, S_I, rc_E) \rightarrow (rc_{I:E})$

$\forall E$: Resolve$(pp, T_E, T_{I:E}, rc_C, rc_{I:E}) \rightarrow (B^E_{final}, B^{I:E}_{final})$

3.2 Bolt Symbols

- Public Parameters (pp) - Public CRS parameters for the zero-knowledge proof scheme.
- Key Pair (pk_x, sk_x) - Public/private encryption key pair belonging to x.
- Balance (B^x_y) - Balance of party x at time y (integer)
- Channel Token (T_x) - Public micropayment channel token. This information should be posted to the ledger in association with the x's escrowed funds.
- Channel Secrets (csk_x) - Secrets associated with a channel.
- Established ($established$) - Boolean signaling successful protocol completion.
- Wallet (w_x)- Encodes the wallet belonging to x. Contains the current channel balance along with information necessary to prove validity of the wallet (along with csk_x).
- Micropayment Value (ϵ_x) - Value of the next payment to be made by x.
- Refund Token (rt_x) - Encodes the information necessary for x to make a claim on channel funds in the event of channel closure.
- Intermediary State (S_I) - Global list maintained by the intermediary that tracks all used (and therefore revealed) wallets to prevent doubles spending.
- Channel Closure Message (rc_x) - Final token that each party submits to the ledger in order to make a claim on channel funds.

3.3 Nanopayment Procedures

Protocol: nInit

Client$(pp, w_c, \delta, n) \rightarrow (nT, ncsk_C, nrt_c, nS_C)$

Intermediary$(pp, S_I, nS_I) \rightarrow (S_I, nS_I)$

Protocol: nEstablish

Client$(nT, ncsk_C) \rightarrow (established)$

Intermediary$(nT, nS_I) \rightarrow (established)$

Relay$(nT, w_R) \rightarrow (nS_R, rt_R)$

Protocol: nPay

Client$(nT, nS_C, ncsk_C) \rightarrow (nS_C)$

Relay$(nT, nS_R) \rightarrow (nS_R)$

$\forall E$: *Protocol*: nClose_E

Relay$(nT, nS_E, rt_E) \rightarrow (w_E)$

Intermediary$(nT, nS_I) \rightarrow (nS_I)$

$\forall E$: nRefund_E$(nT, rt_E, nS_E) \rightarrow (nrc_C)$

$\forall E$: nRefute$(pp, T_E, nS_I, S_I, nrc_E) \rightarrow (nrc_{I:E})$

$\forall E$: nResolve_E$(pp, nT, T_E, T_{I:E}, nrc_{I:E}, nrc_E)$

3.4 Nanopayment Symbols

- Nano Value (δ) - Value of a single nanopayment (all nanopayments values within a channel are uniform).
- Maximum Payments (n) - The maximum number of nanopayments that can be made on the current channel.
- Nano Channel Token (nT) - Defines the setup information about the nanopayment channel. Unlike micropayments, there is only one token that can be freely passed around since privacy is not a concern within the channel scope.
- Nano Channel Secrets ($ncsk_C$) - Secrets owned by the client that allow it to send nanopayments.
- Nano Refund Token (nrt_x) - Identical to a micropayment refund token (rt_x) except that it can be used to claim the latest payment on a nanopayment channel as well.
- Nano State (nS_x) - For endpoint users (clients and relays), the state simply tracks the number of nanopayments that have been made on the single open channel. Since intermediaries have many channels open at once, its nanopayment state holds similar information for all past and present channels as well.
- Nano Channel Closure Message (nrc_x) - Final token that each party submits to claim funds. This is similar to a micropayment channel closure message except that it allows for additional granularity to claim funds within the most recent nanopayment channel.