Cryptographic protocol for playing Risk in an untrusted setting

Jude Southworth

Bachelor of Science in Computer Science and Mathematics
The University of Bath
2023

1 Outline

Risk is a strategy game developed by Albert Lamorisse in 1957. It is a highly competitive game, in which players battle for control over regions of a world map by stationing units within their territories in order to launch attacks on neighbouring territories that are not in their control.

2 Existing solutions

For playing games over an internet connection, multiple solutions already exist. These can roughly be broken down into those that are centralised and those that are decentralised, although many decentralised systems rely on federated or centralised communications for peer discovery.

2.1 Centralised

In highly centralised networks, traffic is routed to a number of servers that are operated by the same organisation who maintains the game or service. This is the current standard for the majority of the internet: in fact, this is the methodology used by the official version of Risk, playable as an app.

Without patching the executables, there is no way for a user to run their own servers, or to connect to a third party's server. This has two main advantages:

- **Moderation.** The developers can enforce their own rules through some form of EULA, and this would be properly enforceable, as if a user is banned from the official servers, there is no alternative.
- **Security.** The server acts as a trusted party, and validates all communications from players. Hence, players cannot subvert a (properly implemented) service's protocol.

2.2 Peer-to-peer networks

In peer-to-peer (P2P) networks, traffic may be routed directly to other peers, or servers may be operated by third parties (sometimes called "federated networks"). This form of communication is still popular in certain games or services, for example BitTorrent is primarily a P2P service; and titles from the Counter-Strike series are federated, with a wide selection of third party hosts.

The main advantage of peer-to-peer networks over centralised networks is longevity. Games such as Unreal Tournament 99 (which is federated) still have playable servers, as the servers are community-run, and so as long as people still wish to play the game, they will remain online (despite the original developers no longer making any profit from the title) (Eatsleeput.com, 2022).

However, security can often be worse in fully peer-to-peer networks than that of fully centralised networks. Peers may send malicious communications, or behave in ways that violate the general rules of the service. As there is no trusted server, there is no easy way to validate communications to prevent peers from cheating.

Some peer-to-peer services try to address issues with security. In file-sharing protocols such as BitTorrent, a tracker supplies hashes of the file pieces to validate the file being downloaded

(Cohen, 2017). However, the downside of this approach is that a trusted party (in this case the tracker) is still required. A malicious tracker could supply bad hashes, or an outdated tracker may expose the peer to security vulnerabilities.

2.3 Untrusted setups

Currently, there exists an online centralised version of the board game Risk.

We aim to apply bit-commitment schemes and zero-knowledge proof protocols to an online P2P variant of Risk, to allow peers to play the game whilst preventing cheating and needing no trusted parties. The variant of interest is the "fog of war" variant, where a player cannot see the unit counts of regions besides those that they own or are neighbouring.

3 Literature review

Centralised systems can securely perform the generation of random values, through using a cryptographically secure random number generator on the server-side, and distributing the values to the clients. This is how dice rolls are processed in centralised online games. However, in a P2P system, something else must be done to simulate the randomness.

For dice rolling, we want that

- No peer can change the probable outcome of the dice (random),
- No peer can deny having rolled the dice (non-repudiation).

We apply the concept of bit commitment schemes to form these guarantees.

3.1 Bit commitment schemes

Bit commitment schemes provide a mechanism for one party to commit to some hidden value and reveal it later. This can be achieved through the use of commutative cryptographic algorithms and with one-way functions.

Commutative cryptography

Shamir, Rivest and Adleman (1981) provides a protocol using bit commitment to play poker. They offer a bit commitment scheme using commutative encryption algorithms based on modular arithmetic. This scheme works by each player encrypting cards, and decrypting in a different order as to obscure the value of the actual cards until all players have decrypted.

Many encryption schemes are not commutative however. One alternative is to use some well-known one-way function, such as SHA, with randomly generated salts.

Bit commitment with one-way functions

Bit commitment schemes can also be implemented using one-way functions:

1. The first party decides on the value m to be committed to.
2. The first party generates some random value r.

3. The first party generates and publishes some value $c = H(m, r)$, where H is an agreed-upon public one-way function.
4. The first party publishes m and r to the second party some time later.
5. The second party computes $c' = H(m, r)$ and validates that $c = c'$.

Blum (1983) provides a protocol for flipping fair coins across a telephone, which is isomorphic to selecting a random value from a set of two values. This cannot be simply repeated though to generate numbers in the range of 1-6, as 6 is not a power of 2.

However, a similar protocol can be used where each player commits to a single value $x \in \mathbb{Z}_6$. As the distribution of outcomes of addition in the group $\mathbb{Z}_n$ is fair, we can then sum the values of x committed to by both players to deduce a final value for the roll. To decrease the amount of communications required for rolling a number of dice, a vector of values can be used.

This protocol relies only on the ability for one party to produce random numbers. We can consider the $\mathbb{Z}_6$-set on $\mathbb{Z}_6$: upon one party selecting $x \in \mathbb{Z}_6$, the other party's selection is from the group $x \cdot \mathbb{Z}_6 = \{x + 0, \ldots, x + 5\} \cong \mathbb{Z}_6$. So, the potential outcomes only require one party to select randomly.

If both parties were to collude and generate non-randomly, this protocol falls through. A potential way around this is to involve other players in the protocol: the same rule applies of only a single player needs to be selecting randomly to produce random outputs. Therefore, so long as there are non-colluding players, this would protect against basic collusion.

3.2 Zero-knowledge proofs

Zero-knowledge proofs form a subset of minimum disclosure proofs, and beyond that, a subset of interactive proofs. Zero-knowledge proofs are defined by three axioms:

- **Completeness.** If the conjecture is true, an honest verifier will be convinced of its truth by a prover.
- **Soundness.** If the conjecture is false, a cheating prover cannot convince an honest verifier (except with some small probability).
- **Zero-knowledge.** This is the condition for a minimum disclosure proof to be considered zero-knowledge. If the conjecture is true, the verifier cannot learn any other information besides the truthfulness.

Zero-knowledge proofs are particularly applicable to the presented problem. They primarily solve two problems:

- The disclosure of some information without leaking other information,
- The proof presented can only be trusted by the verifier, and not by other parties.

We can further formalise the general description of a zero-knowledge proof. Mohr (2007) provides a common formalisation of the concept of a zero-knowledge proof system for a language L by stating that

- For every $x \in L$, the verifier will accept x following interaction with a prover.
- For some polynomial p and any $x \notin S$, the verifier will reject x with probability at least $\frac{1}{p(|x|)}$.

- A verifier can produce a simulator S such that for all $x \in L$, the outputs of $S(x)$ are indistinguishable from a transcript of the proving steps taken with the prover on x.

The final point describes a proof as being *computationally zero-knowledge*. Some stronger conditions exist, which describe the distributions of the outputs of the simulator versus the distributions of the outputs of interaction with the prover.

- **Perfect.** A simulator produced by a verifier produces outputs that are distributed identically to real transcripts.
- **Statistical.** A simulator produced by a verifier gives transcripts distributed identically, except for some constant number of exceptions.

Some proofs described are *honest-verifier* zero-knowledge proofs. In these circumstances, the verifier is required to act in accordance with the protocol for the simulator distribution to behave as expected. We consider verifiers as honest, as it appears they may only impede themselves by acting dishonestly.

Games as graphs

The board used to play Risk can be viewed as an undirected graph. Each region is a node, with edges connecting it to the adjacent regions. For convenience, we also consider the player's hand to be a node, which has all units not in play placed upon it.

Furthermore, the actions taken when playing the game can be seen as constructing new edges on a directed weighted graph. This makes us interested in the ability to prove that the new edges conform to certain rules.

The main game protocol can be considered as the following graph mutations for a player P:

- **Reinforcement.** A player updates the weight on some edges of the graph that lead from the hand node H_P to region nodes $R_1, \ldots, R_n$ in their control.
  - Any adjacent players will then need to undergo proving the number of units on neighbouring regions.
- **Attack.** Player P attacks R_B from R_A. In the event of losing units, the player updates the edge on the graph from R_A to the hand node H_P.

  In the event of winning the attack, the player updates the edge from R_A to R_B to ensure some non-zero amount of units is located in the region.
- **Unit movement.** The player updates an edge from one region R_1 to another neighbouring region R_2.

The goal is then to identify ways to secure this protocol by obscuring the edges and weights, whilst preventing the ability for the player to cheat.

Graphs & ZKPs

Goldreich, Micali and Wigderson (1991) identifies methods to construct zero-knowledge proofs for two graphs being isomorphic or non-isomorphic.

Identifying Risk as a graph therefore enables us to construct isomorphisms as part of the proof protocol. For example, when a player wishes to commit to a movement, it is important to prove

that the initial node and the new node are adjacent. This can be proven by communicating isomorphic graphs, and constructing challenges based on the edges of the original graph.

Adjacency proofs

Proving adjacency of two nodes is akin to proving isomorphism of two graphs. A protocol using challenges could be constructed as follows:

1. The prover commits a new edge between two nodes.
2. The prover constructs an isomorphic graph to the game, and encrypts the edges.
3. The verified challenges either:
    - That the graphs are isomorphic.
    - That the new edge is valid.
4. The prover sends a total decryption key for the graph's nodes, to prove isomorphism to the game board; or a decryption key for the new edge to the isomorphism, to prove adjacency.

These challenges restrict the ability for the prover to cheat: if the two nodes they are committing to are not adjacent, either the prover will need to commit an invalid isomorphism (detected by challenge 1), or lie about the edge they have committed (detected by challenge 2).

Selection between two challenges is the ideal number of challenges to use, as the probability of cheating being detected is $\frac{1}{2}$. Using more challenge options (e.g, n) means the likelihood of the prover cheating a single challenge reduces to $\frac{1}{n}$. This would require much larger numbers of communications to then convince the verifier to the same level of certainty.

Adjacency proofs are necessary to ensure that players move units fairly.

Cheating with negative values

Zerocash is a ledger system that uses zero-knowledge proofs to ensure consistency and prevent cheating. Ledgers are the main existing use case of zero-knowledge proofs, and there are some limited similarities between ledgers and Risk in how they wish to obscure values of tokens within the system.

Publicly-verifiable preprocessing zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) are the building blocks of Zerocash (Ben Sasson et al., 2014), and its successor Zcash. A zk-SNARK consists of three algorithms: `KeyGen`, `Prove`, `Verify`.

These are utilised to construct and verify transactions called `POUR`s. A `POUR` takes, as input, a certain "coin", and splits this coin into multiple outputs whose values are non-negative and sum to the same value as the input. The output coins may also be associated with different wallet addresses.

Zerocash then uses zk-SNARKs as a means to prove that the value of the inputs into a `POUR` is the same as the value of the outputs. This prevents users from generating "debt", or from generating value without going through a minting process (also defined in the Zerocash spec).

Ensuring consistency of weights

A similar issue appears in the proposed system: a cheating player could update the weights on their graph to cause a region to be "in debt". Therefore, we need the protocol to ensure players prove that the sum of all edges is equal to how many units the player has in play (a well-known value).

Additive homomorphic cryptosystems

Some cryptosystems admit an additive homomorphic property: that is, given the public key and two encrypted values $\sigma_1 = E(m_1), \sigma_2 = E(m_2)$, the value $\sigma_1 + \sigma_2 = E(m_1 + m_2)$ is the ciphertext of the underlying operation.

Paillier (1999) defined a cryptosystem based on residuosity classes, which expresses this property. Damgård, Jurik and Nielsen (2010) demonstrates an honest-verifier zero-knowledge proof for proving a given value is 0. Hence, clearly, proving a summation $a + b = v$ can be performed by proving $v - a - b = 0$ in an additive homomorphic cryptosystem.

So, using some such scheme to obscure edge weights should enable verification of the edge values without revealing their actual values.

Reducing communication

In the presented algorithms, interaction is performed fairly constantly, leading to a large number of communications. This will slow the system considerably, and make proofs longer to perform due to network latency.

An alternative general protocol is the Σ-protocol (Groth, 2004). In the Σ-protocol, three communications occur:

- The prover sends the conjecture.
- The verifier sends a random string.
- The prover sends some proofs generated using the random string.

This reduces the number of communications to a constant, even for varying numbers of challenges.

The Fiat-Shamir heuristic (Fiat and Shamir, 1987) provides a method to further reduce communication by constructing non-interactive zero-knowledge proofs using a random oracle. For ledgers, non-interactive zero-knowledge proofs are necessary, as the ledger must be resilient to a user going offline. However, in our case, users should be expected to stay online for an entire session of Risk, and each session is self-contained. So this full transformation is not necessary.

Set membership proofs

Another approach to the problem is to use set membership, which is a widely considered problem in zero-proof literature. In this case, each region would be associated with a set of units from a public "pool" of units. Then, a player needs to prove the cardinality of a set, and the uniqueness/distinctness of its members. A number of constructs exist for analysing and proving in obscured sets.

Accumulators

Defined by Benaloh and de Mare (1994), accumulators form a subset of one-way hash functions that satisfy a *quasi-commutative* property: that is, for some hash function h, $h(h(x_1, y_1), y_2) = h(h(x_1, y_2), y_1)$.

Benaloh and de Mare (1994) also proved that such functions exist, by providing an example based on modular arithmetic. They then used these to construct set membership proofs as follows:

- Take $s_1, \dots, s_n$ a set of users who wish to identify each other, and P_k a public key.
- Each user s_i computes $z = h(h(h(P_k, s_1), \dots), s_n)$ and $z_i = h(h(h(P_k, s_1), \dots), s_n)$ omitting s_i.
- For a user to validate their membership to another user, they publish (z_i, s_i).

Merkle trees

Merkle trees (Merkle, 1988) provide an alternative way of proving set membership, that is more space efficient than accumulators, and doesn't require special hashing functions (any one-way function will work). A Merkle tree stores the hashes of some data in the leaf nodes, and each node above stores the hash of the two nodes below it. The commitment is then the hash of the topmost node.

With this scheme, the data stored in the leaf nodes is totally obscured. However, the constructor of the tree can demonstrate to another user the presence of some data in the tree by revealing the hashes of a subset of the other nodes in the tree. They can also reveal the tree's structure without revealing any contents by revealing all hashes constituting the tree.

Whilst this would be useful in a Risk version in which a player never exposed their unit count, and simply wagered units on an attack; it doesn't apply well to the intended scenario of privately communicating unit counts, as the hash function used is well-known, and so proofs to a single player can easily be replicated by a malicious verifier to other players in the game.

To overcome this issue we want to devise some zero-knowledge system for proving set size. It is then beneficial to consider a public set U containing all of a player's possible units.

Blind signatures

Chaum (1983) describes a process of a blind signature, in which a message is signed without the contents being revealed to the signer. This requires some signing function S which commutes with an encrypting function E, i.e $E^{-1}(S^{-1}(E(m))) = S^{-1}(m)$.

Camenisch, Chaabouni and shelat (2008) demonstrates how blind signatures can be used to construct zero-knowledge set membership proofs for some element σ in a public set Φ, using pairing-based cryptography.

Blind signatures can also be performed with RSA (Bellare et al., 2003). In RSA-based blind signatures, the signing party computes primes p_A, q_A and exponents d, e such that $(m^d)^e \equiv m \mod p_A q_A$. The 2-tuple $(p_A q_A, e)$ is the public key, and is released publicly. The other party computes a random value R, and computes and publishes $B = m \cdot R^e \mod p_A q_A$ for some message m. The signing party then replies with $B^d = (m \cdot R^e)^d \equiv m^d \cdot R \mod p_A q_A$, so that the other party can then extract m^d as R is known only to them. Due to the discrete

logarithm problem, determining the signing key d from this is not computationally feasible. Similarly, it is not feasible for the signer to determine m, as R is not known to them.

RSA blinding can incur a security risk, as by using the same keys to sign and encrypt, a player can be tricked into revealing their private key through a chosen-plaintext attack.

4 Implementation

The implementation provided uses WebSockets as the communication primitive. This is therefore a centralised implementation. However, no verification occurs in the server code, which instead simply "echoes" messages received to all connected clients.

Despite this approach being centralised, it does emulate a fully peer-to-peer environment, and has notable benefits:

- It is faster to develop, use, and test than using a physical system such as mail;
- There is no need for hole-punching or port-forwarding;
- WebSockets are highly flexible in how data is structured and interpreted.

In particular, the final point allows for the use of purely JSON messages, which are readily parsed and processed by the client-side JavaScript.

4.1 Message structure

Messages are given a fixed structure to make processing simpler. Each JSON message holds an `author` field, being the sender's ID; a message ID to prevent replay attacks and associate related messages; and an `action`, which at a high level dictates how each client should process the message.

The action more specifically is one of `ANNOUNCE`, `DISCONNECT`, `KEEPALIVE`, `RANDOM`, and `ACT`. The first three of these are used for managing the network by ensuring peers are aware of each other and know the state of the network. `RANDOM` is designated to be used by the shared-random-value subprotocol defined later. `ACT` is used by players to submit actions for their turn during gameplay.

Each message is also signed to verify the author. This is a standard application of RSA. A hash of the message is taken, then encrypted with the private key. This can be verified with the public key.

RSA keys are accepted by peers on a first-seen basis.

4.2 Paillier

Paillier requires the calculation of two large primes for the generation of public and private key pairs. ECMAScript typically stores integers as floating point numbers, giving precision up to 2^{53}. This is clearly inappropriate for the generation of sufficiently large primes.

In 2020, ECMAScript introduced `BigInt` (TC39, 2020), which are, as described in the spec, "arbitrary precision integers". Whilst this does not hold true in common ECMAScript implementations (such as Chrome's V8), these "big integers" still provide sufficient precision

for the Paillier cryptosystem, given some optimisations and specialisations are made with regards to the Paillier algorithm and in particular the modular exponentiation operation.

It must be noted that `BigInt` is inappropriate for cryptography in practice, due to the possibility of timing attacks as operations are not necessarily constant time (TC39, 2020). In particular, modular exponentiation is non-constant time, and operates frequently on secret data. A savvy attacker may be able to use this to leak information about an adversary's private key.

4.3 Modular exponentiation

As `BigInt`'s V8 implementation does not optimise modular exponentiation, we employ the use of addition chaining, as described in Schneier (1996). Addition chaining breaks a modular exponentiation into repeated square-and-modulo operations, which are computationally inexpensive to perform.

The number of operations is dependent primarily on the size of the exponent. For an exponent of bit length L, somewhere between L and $2L$ multiply-and-modulo operations are performed, which gives overall a logarithmic time complexity supposing bit-shifts and multiply-and-modulo are constant time operations.

4.4 Generating large primes

I chose to use primes of length 2048 bits. This is a typical prime size for public-key cryptography, as this generates a modulus $n = pq$ of length 4096 bits.

Generating these primes is a basic application of the Rabin-Miller primality test (Rabin, 1980). This produces probabilistic primes, however upon completing sufficiently many rounds of verification, the likelihood of these numbers actually not being prime is dwarfed by the likelihood of hardware failure.

4.5 Public key

In the Paillier cryptosystem, the public key is a pair (n, g) where $n = pq$ for primes p, q satisfying $\gcd(pq, (p-1)(q-1)) = 1$ and $g \in \mathbb{Z}^*_{n^2}$. We restrict the range of plaintexts m to $m < n$.

The Paillier cryptosystem is otherwise generic over the choice of primes p, q. However, by choosing p, q of equal length, the required property on $pq, (p-1)(q-1)$ coprime is guaranteed.

Proposition 4.1. *For p, q prime of equal length,* $\gcd(pq, (p-1)(q-1)) = 1$.

Proof. Without loss of generality, assume $p > q$. Suppose $\gcd(pq, (p-1)(q-1)) \neq 1$. Then, $q \mid p-1$. However, the bit-lengths of p, q are identical. So $\frac{1}{2}(p-1) < q$. This is a contradiction to $q \mid p-1$ (as 2 is the smallest possible divisor), and so we must have $\gcd(pq, (p-1)(q-1)) = 1$ as required. □

As the prime generation routine generates primes of equal length, this property is therefore guaranteed. The next optimisation is to select $g = 1 + n$.

Proposition 4.2. $1 + n \in \mathbb{Z}^*_{n^2}$.

Proof. We see that $(1+n)^n \equiv 1 \mod n^2$ from binomial expansion. So $1+n$ is invertible as required. □

The selection of such g is ideal, as the binomial expansion property helps to optimise exponentiation. Clearly, from the same result, $g^m = 1 + mn$. This operation is far easier to perform, as it can be performed without having to take the modulus to keep the computed value within range.

4.6 Encryption

The ciphertext is, in general, computed as $c = g^m r^n \mod n^2$ for $r < n$ some random secret value. To make this easier to compute, we compute the equivalent value $c = (r^n \mod n^2) \cdot (g^m \mod n^2) \mod n^2$.

4.7 Private key

The private key is the value of the Carmichael function $\lambda = \lambda(n)$, defined as the exponent of the group $\mathbb{Z}_n^*$. From the Chinese remainder theorem, $\lambda(n) = \lambda(pq)$ can be computed as $\text{lcm}(\lambda(p), \lambda(q))$. From Carmichael's theorem, this is equivalent to $\text{lcm}(\phi(p), \phi(q))$, where ϕ is Euler's totient function. Hence, from the definition of Euler's totient function, and as p, q are equal length, $\lambda = (p-1)(q-1) = \phi(n)$.

We are also interested in the ability to compute $\mu = \lambda^{-1} \mod n$ as part of decryption. Fortunately, this is easy, as from Euler's theorem, $\lambda^{\phi(n)} \equiv 1 \mod n$, and so we propose $\mu = \lambda^{\phi(n)-1} \mod n$. As $\phi(n)$ is well-known to us, we get $\mu = \lambda^{(p-1)(q-1)} \mod n$, a relatively straight-forward computation.

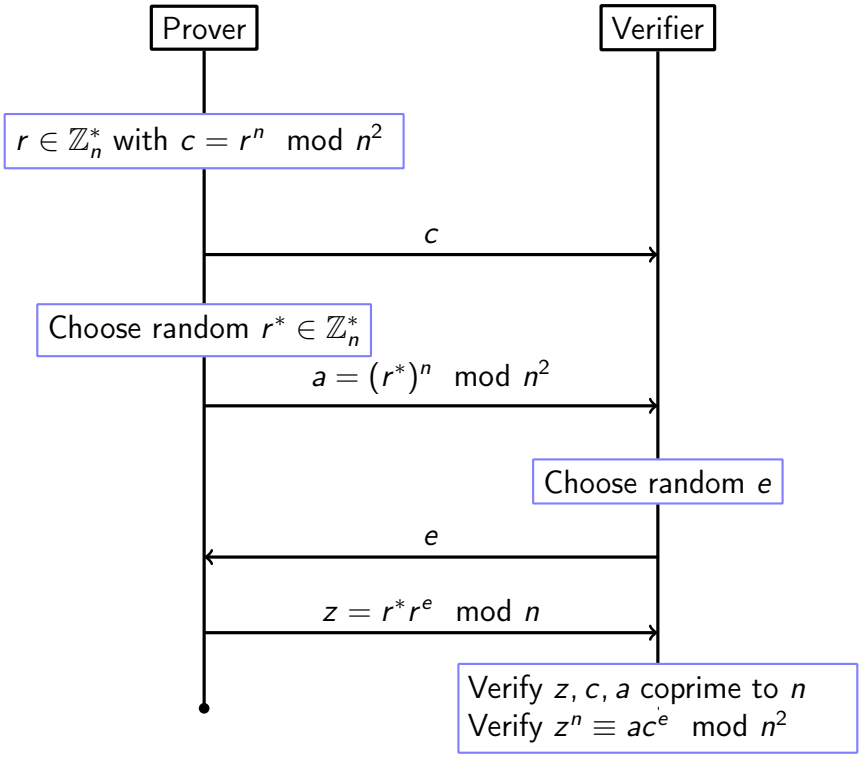
4.8 Decryption

Let c be the ciphertext. The corresponding plaintext is computed as $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$, where $L(x) = \frac{x-1}{n}$. This is relatively simple to compute in JavaScript.

4.9 Proof system

The proof system is that of Damgård, Jurik and Nielsen (2003). The authors give a method to prove knowledge of an encrypted value. The importance of using a zero-knowledge method for this is that it verifies knowledge to a single party. This party should be an honest verifier: this is an assumption we have made of the context, but in general this is not true, and so this provides an attack surface for colluding parties.

The proof system presented is an interactive proof for a given ciphertext c being an encryption of 0.



A proof for the following homologous problem can be trivially constructed: given some ciphertext $c = g^m r^n \mod n^2$, prove that the text $cg^{-m} \mod n^2$ is an encryption of 0. The text cg^{-m} is constructed by the verifier. The prover then proceeds with the proof as normal, since cg^{-m} is an encryption of 0 under the same noise as the encryption of m given.

4.10 Implementation details

4.11 Application to domain

Players should prove a number of properties of their game state to each other to ensure fair play. These are as follows.

1. The number of reinforcements placed during the first stage of a turn.
2. The number of units on a region neighbouring another player.
3. The number of units lost during an attack/defence.
4. The number of units available for an attack/defence.
5. The number of units moved when fortifying.

(4) and (5) can be generalised further as range proofs.

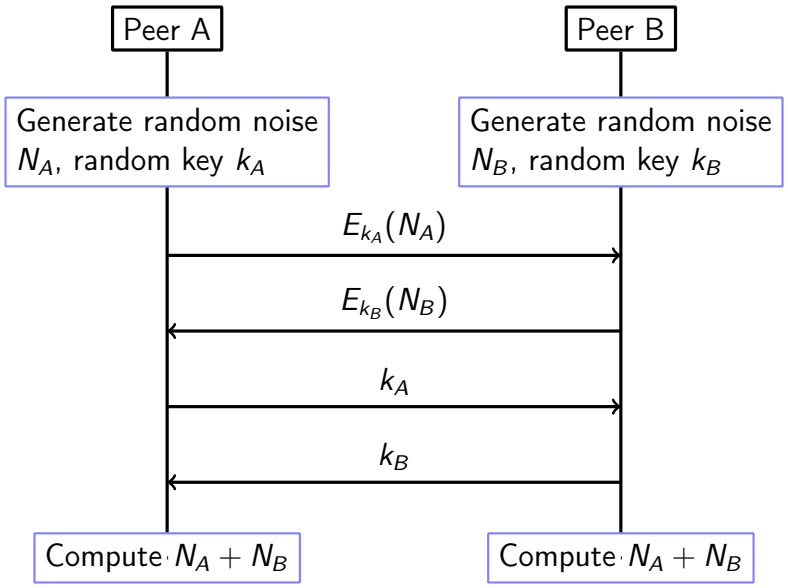
For (1), we propose the following communication sequence. The player submits pairs (R, c_R) for each region they control, where R is the region and c_R is a ciphertext encoding the number of reinforcements to add to the region (which may be 0). Each player computes $c_{R_1} \cdot \ldots \cdot c_{R_n}$.

4.12 Shared random values

A large part of Risk involves random behaviour dictated by rolling some number of dice. To achieve this, some fair protocol must be used to generate random values consistently across

each peer without any peer being able to manipulate the outcomes.

This is achieved through bit-commitment and properties of $\mathbb{Z}_n$. The protocol for two peers is as follows, and generalises to n peers trivially.



Depending on how $N_A + N_B$ is then turned into a random value within a range, this system may be manipulated by an attacker who has some knowledge of how participants are generating their noise. As a basic example, suppose a random value within range is generated by taking $N_A + N_B \mod 3$, and participants are producing 2-bit noises. An attacker could submit a 3-bit noise with the most-significant bit set, in which case the odds of getting a 1 are significantly higher than the odds of a 0 or a 2. To avoid this problem, peers should agree beforehand on the number of bits to transmit, and truncate any values in the final stage that exceed this limit.

The encryption function used must also guarantee the integrity of decrypted ciphertexts to prevent a malicious party creating a ciphertext which decrypts to multiple valid values through using different keys.

Proposition 4.3. *The scheme shown is not manipulable by a single cheater.*

Proof. Suppose $P_1, \ldots, P_{n-1}$ are honest participants, and P_n is a cheater with desired outcome O.

The encryption function E_k holds the confidentiality property: that is, without k, P_i cannot retrieve m given $E_k(m)$.

Each participant P_i commits N_i. Then, the final value is $N_1 + \cdots + N_{n-1} + N_n$. □

4.13 Avoiding modular bias

The typical way to avoid modular bias is by resampling. To avoid excessive communication, resampling can be performed within the bit sequence by partitioning into blocks of n bits and taking blocks until one falls within range. This is appropriate in the presented use case as random values need only be up to 6, so the probability of consuming over 63 bits of noise when resampling for a value in the range 0 to 5 is $\left(\frac{1}{4}\right)^{21} \approx 2.3 \times 10^{-13}$.

Bibliography

Bellare, M., Namprempre, C., Pointcheval, D. and Semanko, M., 2003. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of cryptology*, 16(3).

Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E. and Virza, M., 2014. Zerocash: Decentralized anonymous payments from bitcoin [Online]. *2014 ieee symposium on security and privacy*. pp.459–474. Available from: `https://doi.org/10.1109/SP.2014.36`.

Benaloh, J. and Mare, M. de, 1994. One-way accumulators: A decentralized alternative to digital signatures. In: T. Helleseth, ed. *Advances in cryptology — eurocrypt '93*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.274–285.

Blum, M., 1983. Coin flipping by telephone a protocol for solving impossible problems. *Acm sigact news*, 15(1), pp.23–27.

Camenisch, J., Chaabouni, R. and shelat, a., 2008. Efficient protocols for set membership and range proofs. In: J. Pieprzyk, ed. *Advances in cryptology - asiacrypt 2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.234–252.

Chaum, D., 1983. Blind signatures for untraceable payments. In: D. Chaum, R.L. Rivest and A.T. Sherman, eds. *Advances in cryptology*. Boston, MA: Springer US, pp.199–203.

Cohen, B., 2017. Bittorrent.org. Available from: `https://www.bittorrent.org/beps/bep_0003.html`.

Damgård, I., Jurik, M. and Nielsen, J.B., 2010. A generalization of paillier's public-key system with applications to electronic voting. *International journal of information security*, 9(6), pp.371–385.

Damgård, I., Jurik, M. and Nielsen, J., 2003. A generalization of paillier's public-key system with applications to electronic voting. *International journal of information security* [Online], 9, pp.371–385. Available from: `https://doi.org/10.1007/s10207-010-0119-9`.

Eatsleeput.com, 2022. [Online]. Archive: https://archive.ph/Gp0Ou. Available from: `https://eatsleeput.com/`.

Fiat, A. and Shamir, A., 1987. How to prove yourself: Practical solutions to identification and signature problems. In: A.M. Odlyzko, ed. *Advances in cryptology — crypto' 86*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.186–194.

Goldreich, O., Micali, S. and Wigderson, A., 1991. Proofs that yield nothing but their validity

or all languages in np have zero-knowledge proof systems. *J. acm* [Online], 38(3), p.690–728. Available from: https://doi.org/10.1145/116825.116852.

Groth, J., 2004. *Honest verifier zero-knowledge arguments applied*. Ph.D. thesis. BRICS.

Merkle, R.C., 1988. A digital signature based on a conventional encryption function. In: C. Pomerance, ed. *Advances in cryptology — crypto '87*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.369–378.

Mohr, A., 2007. A survey of zero-knowledge proofs with applications to cryptography. *Southern illinois university, carbondale*, pp.1–12.

Paillier, P., 1999. Public-key cryptosystems based on composite degree residuosity classes. *International conference on the theory and applications of cryptographic techniques*. Springer, pp.223–238.

Rabin, M.O., 1980. Probabilistic algorithm for testing primality. *Journal of number theory* [Online], 12(1), pp.128–138. Available from: https://doi.org/https://doi.org/10.1016/0022-314X(80)90084-0.

Schneier, B., 1996. *Applied cryptography*. John Wiley.

Shamir, A., Rivest, R.L. and Adleman, L.M., 1981. *Mental poker* [Online], Boston, MA: Springer US, pp.37–43. Available from: https://doi.org/10.1007/978-1-4684-6686-7_5.

TC39, 2020. Bigint: Arbitrary precision integers in javascript. https://github.com/tc39/proposal-bigint.