BitcoinBT (BTCBT)

Technical Specification

I. Consensus Structure and Fork Boundary Definition

1. Consensus Inheritance and Fork Activation Model
    - 1.1 Definition of Chain Inheritance Scope
    - 1.2 Definition of Fork Boundary Constants
    - 1.3 Branch Function $R(h)$
    - 1.4 Height Computation Rule
    - 1.5 Consensus Function Dispatch Path
    - 1.6 State Transition Structure
    - 1.7 Chain Selection Principle Based on Accumulated Work

II. Monetary Policy and Supply Cap Structure

2. Definition of the Block Subsidy Function
    - 2.1 Initial Subsidy Constant
    - 2.2 Halving Interval
    - 2.3 Integer-Shift-Based Subsidy
3. Proof of the Total Supply Cap
4. 630,000 BTCBT Redistribution Structure
    - 4.1 Redistribution Block
    - 4.2 Definition of the Decrement Interval
    - 4.3 Proof of Net-Supply Invariance

III. Difficulty Adjustment and Time Model

5. ASERT Difficulty Adjustment
6. Timestamp Validation Conditions

IV. Block and Header Validation

7. Header Validation
    - 7.1 Proof-of-Work Condition
    - 7.2 Target Reconstruction Process
    - 7.3 Compact Representation Consistency
8. Block Structure Validation

8.1 Merkle Root Integrity
8.2 Maximum Block Size
8.3 Block Weight Upper Bound
8.4 SigOps Cost Limit

V. State Transition and Consensus Application

9. Contextual Consensus Validation

10. UTXO State Transition Model

VI. Chain Selection and Security Model

11. Accumulated Work Computation

12. Reorg and Attack Model

VII. Consensus Parameters and Change Scope

13. Definition of Consensus Parameter Constants

14. Unchanged Elements

15. Changed Elements

VIII. Summary of Consensus Invariants

16. Consensus Invariants

Technical Specification(기술명세서KR)

I. Consensus Structure and Fork Boundary Definition

1. Consensus Inheritance and Fork Activation Model

1.1 Definition of Chain Inheritance Scope

BitcoinBT (BTCBT) deterministically inherits the consensus state of the Bitcoin main chain up to block height **903,844**.
Here, "inheritance" does not mean copying block data; it means reproducing identical state transitions by applying identical consensus rules.

The inherited target elements are as follows:
· 80-byte block header structure
· SHA-256d proof-of-work structure

· UTXO state transition results
· Script execution semantics
· SegWit and Taproot validation paths
· Chain selection rule based on nChainWork

All blocks up to height 903,844 are re-validated under the same consensus rules, and as a result, the derived UTXO set, accumulated work, and MedianTimePast values are formed identically to Bitcoin.
This is inheritance based on consensus revalidation, not snapshot copying.

In the pre-fork segment, the following are not changed:
· Script semantics
· Transaction serialization format
· Block header format
· State transition model

The changed elements are limited to the post-fork difficulty computation method and certain consensus constants.

1.2 Definition of Fork Boundary Constants

The fork boundary is defined by the following constant:

$$H_f = 903{,}844$$

This value is a hard-coded integer constant in the chain parameters and is not changed by runtime configuration, external signaling, or network voting.

The first BTCBT block after the fork is defined as:

$$H_a = H_f + 1 = 903{,}845$$

There is no intermediate transition segment.

1.3 Definition of the Branch Function $R(h)$

Consensus branching is defined as a height-based decision function.

$$R(h) = \begin{cases} R_{BTC} & \text{if } h \leq H_f \\ R_{BTCBT} & \text{if } h \geq H_a \end{cases}$$

This function is determined by the single constant H_f and does not include probabilistic

factors or state-dependent conditions.
For the same block data, all nodes select the same $R(h)$.

1.4 Block Height Computation Rule

Block height is not stored in an internal block field; it is computed through the chain index.

$$h = h_{parent} + 1$$

This definition guarantees the following:
· The height cannot be forged via manipulation of internal block data
· Off-by-one errors at the fork boundary are eliminated
· Discrepancies in height computation among nodes are eliminated

Consensus switching is determined by chain context, not by external input.

1.5 Consensus Function Dispatch Structure

The consensus differences before and after the fork exist in the set of validation rules, not in the state model.

$$h \leq H_f \rightarrow \text{Legacy Difficulty Path}$$
$$h \geq H_a \rightarrow \text{ASERT Difficulty Path}$$

Elements that are preserved:
· UTXO model
· Script semantics
· Transaction serialization
· Block header structure
· Accumulated work computation method

Changed elements:
· Difficulty adjustment formula
· Certain consensus parameter values

The state transition structure itself is not changed.

1.6 State Transition Model

BTCBT maintains the same UTXO-based state model as Bitcoin.
The state is defined as:

$$State_h = (UTXO_h, ChainWork_h, MTP_h)$$

where:

· $UTXO_h$: set of unspent outputs at height h

· $ChainWork_h$: accumulated work

· MTP_h: MedianTimePast

MTP is computed as the median timestamp of the most recent 11 blocks.
Block application is defined as:

$$State_h = Apply(Block_h, State_{h-1})$$

Conditions:

· The state is updated only if block validation succeeds

· If validation fails, the state is preserved atomically

· Consensus judgment depends only on $(Block_h, State_{h-1})$

The fork does not change the state structure.

1.7 Chain Selection Based on Accumulated Work

The work of an individual block is defined as:

$$Work(h) = \left\lfloor \frac{2^{256}}{Target(h) + 1} \right\rfloor$$

The purpose of the $+1$term is as follows:

· Preventing the Target = 0 exception

· Ensuring integer boundary stability

Accumulated work is:

$$ChainWork(h) = ChainWork(h-1) + Work(h)$$

The active chain is defined as the chain with maximum accumulated work:

$$ActiveChain = \arg\max(ChainWork)$$

The selection criterion is not the number of blocks, but computed work.
This principle is not changed after the fork.

II. Monetary Policy and Supply Cap Structure

2. Definition of the Block Subsidy Function

2.1 Initial Subsidy Constant

The initial block subsidy is defined by the following constant:

$$S_0 = 50 \text{ BTCBT}$$

All reward computations are performed using integers down to the minimum unit (10^{-8} BTCBT, satoshi unit).
Consensus arithmetic is processed using integer arithmetic only, and floating-point arithmetic is not used.

This structure guarantees the following:
· Complete reproducibility of computed results across platforms
· Exclusion of rounding error
· Prevention of consensus divergence among nodes
· Deterministic stability of long-term supply computation

The subsidy is computed within the range of 64-bit integers under consensus rules.

2.2 Halving Interval

The halving interval is defined as:

$$I = 210{,}000$$

Since the target block interval is 300 seconds, the time length of a halving period is:

$$T_{halving} = 210{,}000 \times 300 = 63{,}000{,}000 \text{ seconds} \approx 2 \text{ years}$$

The halving step is defined for block height $h$as:

$$k = \left\lfloor \frac{h}{I} \right\rfloor$$

where:
· h: block height
· k: halving step index

The reasons for using the integer floor operation are as follows:
· Each block belongs to exactly one halving interval
· Preventing reward overlap or gaps at boundary blocks
· Eliminating computational variation across implementation environments

Halving intervals are discrete, and there is no overlap of contiguous segments.

2.3 Integer-Shift-Based Subsidy Computation

The base block subsidy is defined as:

$$BaseSubsidy(h) = S_0 \gg k$$

This is mathematically equivalent to:

$$BaseSubsidy(h) = \frac{S_0}{2^k}$$

The reasons for using bit-shift operations are:
· Exact implementation of division by powers of two
· Excluding floating-point usage
· Guaranteeing complete determinism of consensus computation

The subsidy value always satisfies the following condition:

$$BaseSubsidy(h) \geq 0$$

After the point at which the subsidy becomes less than 1 satoshi, the subsidy value is fixed at 0.
Negative subsidy is not permitted at the consensus level.

2.4 Total Supply Convergence Structure

Block subsidy decreases geometrically according to the halving structure.
The per-block subsidy at halving step $k$is:

$$S_k = \frac{S_0}{2^k}$$

The number of blocks in each halving interval is always fixed:

$$I = 210{,}000$$

Therefore, the total subsidy issued in the k-th interval is:

$$Emission_k = I \times \left(\frac{S_0}{2^k}\right)$$

Mathematical expression of total issuance

Total issuance is expressed as the following infinite series:

$$Total = \sum_{k=0}^{\infty} I \times \left(\frac{S_0}{2^k}\right)$$

Rearranging the constant factors:

$$Total = I \times S_0 \times \sum_{k=0}^{\infty} \left(\frac{1}{2^k}\right)$$

By the geometric-series sum formula:

$$\sum_{k=0}^{\infty} \left(\frac{1}{2^k}\right) = 2$$

Therefore, the theoretical total supply cap is:

$$Total_{max} = 2 \times I \times S_0$$

Substituting the initial constants:

$$Total_{max} = 2 \times 210{,}000 \times 50 = 21{,}000{,}000 \text{ BTCBT}$$

2.5 Actual Convergence Property Under Integer Halving

Because the consensus logic operates based on integer shifting, actual computation is performed as follows:

$$BaseSubsidy(h) = S_0 \gg k$$

In an integer-shift structure, subsidy becomes 0 after a certain point:

$$BaseSubsidy(h) = 0 \text{when} \frac{S_0}{2^k} < 1 \text{ satoshi}$$

Therefore, actual total supply converges asymptotically to the mathematical upper bound and does not exceed it.
Consensus invariant:

$$Total_{emitted} \leq 21{,}000{,}000 \text{ BTCBT}$$

This condition is enforced by the halving structure and integer-shift operations.

2.6 Consensus Meaning of the Supply Cap

The supply cap is not a parameter but a result derived from the computation structure.
· The halving interval $I$is constant
· The initial subsidy $S_0$is constant
· The decay rate is exactly 1/2
· Computation is integer-based

As long as these four conditions are satisfied, total supply cannot structurally exceed 21,000,000 BTCBT.
This is a mathematical consequence independent of external governance or policy changes.

3. Proof of the Total Supply Cap

Total issuance is expressed as the sum of a geometric series based on the halving structure:

$$Total = \sum_{k=0}^{\infty} \left(S_0 \cdot \frac{1}{2^k} \cdot I \right)$$

Grouping constant factors:

$$Total = S_0 \cdot I \cdot \sum_{k=0}^{\infty} \left(\frac{1}{2^k} \right)$$

The geometric series converges as:

$$\sum_{k=0}^{\infty}\left(\frac{1}{2^k}\right) = 2$$

Therefore, the theoretical total issuance cap is:

$$Total = 50 \cdot 210{,}000 \cdot 2 = 21{,}000{,}000 \text{ BTCBT}$$

This is the supply cap value derived directly from the mathematical halving structure.

3.1 Actual Supply Properties Under Integer-Based Implementation

In the consensus implementation, subsidy is computed by integer shifting:

$$BaseSubsidy(h) = S_0 \gg k$$

This operation truncates fractional parts.
Therefore, fine-grained units generated at each halving step do not accumulate.
As a result, actual cumulative supply always satisfies:

$$TotalSupply \leq 21{,}000{,}000 \text{ BTCBT}$$

3.2 Consensus Meaning

This inequality implies the following:
· Issuance exceeding the cap does not occur under the consensus structure
· Additional issuance requires changing the consensus code itself
· As long as network consensus is maintained, issuance exceeding 21,000,000 BTCBT is structurally impossible

The supply cap is not a policy declaration; it is a consensus invariant derived from the halving formula and integer arithmetic structure.

4. 630,000 BTCBT Redistribution Structure

This structure is not an inflation mechanism that creates new supply; it is a time redistribution within the total supply cap (21,000,000 BTCBT).
That is, after paying the reward in advance at a specific block, the same quantity is decremented across a subsequent interval so that the net change in total issuance remains 0.
The total supply cap is not changed.

4.1 Definition of the Redistribution Block

The block height at which redistribution is applied is as follows:

$$H_r = 903{,}850$$

The subsidy at this block is defined as:

$$Subsidy(H_r) = BaseSubsidy(H_r) + 630{,}000$$

where:

· $BaseSubsidy(H_r)$: base subsidy under halving rules

· 630,000 BTCBT: redistributed amount

The additional subsidy applies only at a single block.

Consensus conditions:

· Coinbase outputs must not exceed the above definition.

· If exceeded, the block is invalid under consensus rules.

This condition is enforced at the consensus layer, not the policy layer.

4.2 Definition of the Decrement Interval

The number of decrement blocks to offset the redistributed amount is:

$$N = 630{,}000$$

In the interval after the redistribution block, subsidy is adjusted as:

$$Subsidy(h) = BaseSubsidy(h) - 1$$

However, the following condition is a prerequisite:

$$BaseSubsidy(h) \geq 1$$

Therefore:

· Decrement applies only while subsidy is at least 1 BTCBT

· Decrement stops after subsidy converges to 0

· Negative subsidy is not permitted under consensus

Total decrement amount is:

$$TotalDecrease = N \times 1 = 630{,}000$$

4.3 Proof of Net-Supply Invariance

Total increase due to redistribution:

$$+630{,}000$$

Total decrement afterward:

$$-630{,}000$$

Net change:

$$\Delta Supply = +630{,}000 - 630{,}000 = 0$$

Therefore:

$$TotalSupply_{after} = TotalSupply_{before}$$

This means the following:
· Long-term total issuance does not change
· The 21,000,000 BTCBT cap is maintained
· Redistribution adjusts the time distribution of issuance
· It is not an inflation or additional supply mechanism

The total supply cap is maintained at the consensus level by the combination of the halving structure and the decrement offset mechanism.

III. Difficulty Adjustment and Time Model

5. ASERT Difficulty Adjustment

5.1 Definition of the Anchor Reference Block

ASERT (Absolutely Scheduled Exponentially Weighted Target) uses the first block after the fork as the anchor.

$$h_0 = 903{,}845$$

At the anchor block, the following values are used as fixed references:
· T_0: Target of the anchor block
· t_0: timestamp of the anchor block

All block difficulties after the anchor are computed solely from these two values and the current block information.
External state or historical window averaging is not used.

5.2 Time Error Computation

At height $h$after the anchor, time deviation is defined as:

$$\Delta t = (t_h - t_0) - 300(h - h_0)$$

where:
· 300: target block interval (seconds)
· $(h-h_0)$: number of blocks elapsed since anchor
· (t_h-t_0): actual elapsed time

Meaning of Δt:
· $\Delta t > 0 \rightarrow$ actual block production is slower than target
· $\Delta t < 0 \rightarrow$ actual block production is faster than target

Δtrepresents the difference between "actual cumulative time" and "theoretical cumulative time."

5.3 Exponential Correction Formula

ASERT difficulty is defined as:

$$Target(h) = T_0 \cdot 2^{\Delta t/172{,}800}$$

where:
· 172,800 seconds = half-life constant (2 days)

Meaning of half-life:
· $\Delta t = +172{,}800$seconds $\rightarrow$ $Target = 2 \cdot T_0$
· $\Delta t = -172{,}800$seconds $\rightarrow$ $Target = 0.5 \cdot T_0$

That is, time deviation is corrected continuously in an exponential manner.
This structure guarantees:
· Removal of fixed-length adjustment windows

· Removal of difficulty-gap segments
· Immediate response to abrupt hashrate changes
· Long-term elimination of cumulative time error

ASERT is a per-block continuous adjustment model.

5.4 powLimit Upper Bound

Difficulty satisfies the following upper bound:

$$Target(h) \leq powLimit$$

powLimit means the maximum permitted Target (minimum difficulty).
This condition prevents difficulty from becoming arbitrarily low and limits the consensus-permitted Target range.

5.5 Consensus Consistency of nBits

The difficulty representation included in the block header is compact format (nBits).
Consensus condition:

$$Expand(nBits_h) = Target(h)$$

That is, the difficulty recorded in the header must match the ASERT computation result exactly.
If inconsistent:
· the block is rejected at the consensus level
· it is impossible for a miner to declare an arbitrary difficulty

5.6 Consensus Constraints on Timestamp

Because ASERT depends on timestamp, time values are subject to consensus constraints:

$$Timestamp(h) > MTP(h-1)$$
$$Timestamp(h) \leq NetworkTime + 7200$$

where:
· $MTP(h-1)$: median timestamp of the most recent 11 blocks
· 7200 seconds: maximum permitted future drift (2 hours)

These constraints guarantee:
· prevention of backward time movement into the past

· limiting difficulty manipulation via future-time distortion
· stability of Δtcomputation

Timestamp influences difficulty computation, but structural distortion is blocked at the consensus level by these constraints.

6. Timestamp Validation Conditions

A block's timestamp directly affects difficulty computation (ASERT) and chain ordering. Therefore, timestamp is subject to explicit constraints at the consensus level.
BTCBT maintains the same time-validation structure as Bitcoin Core, and these constraints are not changed after ASERT adoption.

6.1 MedianTimePast Constraint

Each block $h$must satisfy:

$$Timestamp(h) > MTP(h-1)$$

MedianTimePast is defined as:

$$MTP(h-1) = Median(Timestamp_{h-1}, \dots, Timestamp_{h-11})$$

That is, it uses the median of the previous 11 block timestamps.

This structure guarantees:
· prevention of timestamp backdating
· blocking past-time manipulation by a single block
· limiting abrupt negative distortion of time deviation in difficulty computation

The reason for using the median is that it is less sensitive to outliers than the mean. Therefore, even if some blocks contain abnormal timestamps, the consensus reference time is not distorted abruptly.

6.2 Future-Time Upper Bound

Block timestamp must satisfy the following upper bound:

$$Timestamp(h) \leq NetworkTime + 7200$$

where:
· NetworkTime: network-synchronized time

· 7200 seconds: maximum permitted future drift (2 hours)

This constraint guarantees:
· prevention of excessive future timestamps
· blocking artificial time acceleration intended to force difficulty drops
· suppression of long-term accumulation of time distortion

7200 seconds is a consensus safety bound considering typical network time error ranges.

6.3 Interaction Between ASERT and Time Error

ASERT difficulty is determined by the following variable:

$$\Delta t = (t_h - t_0) - 300(h - h_0)$$

Timestamp manipulation can theoretically affect Δt.
However, the following three conditions combine to constrain structural distortion:

1. MTP constraint
   → abrupt movement into the past is not possible
2. Future bound (7200 seconds)
   → large future-time declarations are not possible
3. Continuous exponential correction structure of ASERT
   → if time deviation accumulates, difficulty responds immediately

Because ASERT is not a fixed-window model, if time error accumulates beyond a certain level, difficulty is continuously adjusted.

6.4 Consensus Outcome

Due to the above structure, the following holds:
· timestamps influence difficulty but unlimited manipulation is not possible
· short-term time deviations are mitigated by the median structure
· persistent time distortion is offset by difficulty adjustment

Timestamp is an input to difficulty computation, but consensus constraints structurally maintain chain stability.

IV. Block Header Validation

Block header validation is performed prior to transaction validation.
This corresponds to the first stage of consensus validation, where proof-of-work and difficulty conditions are checked first.

A block that fails header validation does not proceed to subsequent stages and is immediately invalidated.
The purpose of header validation is as follows:
· early blocking of invalid PoW blocks
· preventing expensive transaction-validation computation
· performing preliminary consensus checks with minimal computation

7.1 Proof-of-Work Condition

The block header hash must satisfy:

$$Hash_{SHA256d}(Header_h) \leq Target(h)$$

where:
· $Hash_{SHA256d}$: double-SHA256 hash of the block header (256-bit integer)
· $Target(h)$: target computed by ASERT

Target is the maximum permitted hash value.
A block is valid only when its hash is less than or equal to Target.

The relationship between Difficulty and Target is:

$$Difficulty \propto \frac{1}{Target}$$

As Target decreases, the expected computational work increases.

All comparisons are performed within 256-bit integer range.
Floating-point arithmetic is not used.
A block that does not satisfy the condition is immediately rejected.

7.2 Target Reconstruction and Consistency Validation

The block header contains the compact-format difficulty representation nBits.
Consensus enforces:

$$Expand(nBits_h) = Target(h)$$

Expand(·) is a function that converts the compact format into a 256-bit integer Target.

The reconstruction procedure consists of the following steps:

1. Split nBits into exponent and mantissa
2. Expand into a normalized 256-bit integer form
3. Perform overflow and range checks

The meaning of this condition is as follows:
· a miner cannot declare an arbitrary Target
· complete equality between the header value and the ASERT-computed value is enforced
· difficulty tampering is blocked

If the ASERT computation result and $Expand(nBits_h)$do not match, the block is invalid under consensus.

7.3 Compact Representation Consistency Constraints

Target must satisfy the following range:

$$0 < Target(h) \leq powLimit$$

The following conditions are also verified:
· Target cannot be negative
· overflow cannot occur
· Target cannot exceed powLimit

This check is performed before the PoW comparison.
Its purposes are:
· blocking abnormal difficulty encodings
· preventing mathematical range violations
· preventing consensus-level difficulty manipulation

8. Block Structure Validation

After passing header validation, structural validation of the block body is performed.
At this stage, the following consensus constants are validated:

· data integrity
· serialized block-size limit
· weight computation
· signature-operation cost limit

8.1 Merkle Root Integrity

Let the transaction set in the block be $TxSet_h$. The following must hold:

$$MerkleRoot(TxSet_h) = Header.MerkleRoot$$

That is:
· the computed result of the hash tree over the transaction set must match the MerkleRoot recorded in the header exactly.

The Merkle tree follows a double-SHA256-based hash-tree structure.
This condition guarantees:
· transactions cannot be added
· transactions cannot be removed
· transaction order cannot be changed

If inconsistent, the block is immediately invalidated.

8.2 Serialized Block Size Limit

Serialized block size must satisfy:

$$SerializedBlockSize \leq 32{,}000{,}000 \text{ bytes}$$

This is a consensus constant.
Because it is enforced at the consensus level (not the policy level), blocks exceeding it are rejected across the network.

The purposes of this upper bound are:
· ensuring network propagation stability
· maintaining an upper bound on node resource usage
· clarifying the throughput expansion range

8.3 Block Weight Computation and Upper Bound

Block Weight is defined as:

$$Weight = BaseSize \times 4 + WitnessSize$$

where:
· BaseSize: non-witness data size
· WitnessSize: witness data size

This definition matches the SegWit structure.

Upper-bound structure
The non-contextual baseline constant is:

$$MAX\_BLOCK\_WEIGHT = 4{,}000{,}000$$

However, after the BTCBT fork, the actual upper bound is determined by consensus.btcbt_max_block_size.
Therefore, the consensus condition is expressed as:

$$Weight \leq ContextualMaxBlockWeight(h)$$

Pre-fork:

$$Weight \leq 4{,}000{,}000$$

Post-fork:

$$Weight \leq btcbt\_max\_block\_size \times 4$$

(If btcbt_max_block_size = 32MB, the maximum is 128,000,000 weight.)

Through this:
· the existing Bitcoin validation structure is preserved
· the SegWit weight model is preserved
· an expanded upper bound is applied after the fork

Blocks exceeding the bound are invalid under consensus.

8.4 SigOps Cost Limit

Total signature-verification cost in a block must satisfy:

$$SigOpsCost \leq 80{,}000$$

SigOpsCost is the sum of signature verification operation costs incurred during script execution for all transactions in the block.

The purposes of this upper bound are:
· preventing excessive computation via script
· ensuring predictability of validation time
· maintaining a consensus-level computation upper bound

Blocks exceeding this upper bound are invalid under consensus.

V. Contextual Consensus Validation

Contextual consensus validation selects the applicable consensus rules according to a block's chain position (height).
Even with identical block structure, difficulty computation and subsidy rules can differ by height.
This selection is determined based on height computed from the chain index.

9.1 Difficulty Path Switching

The difficulty computation algorithm branches at the fork boundary as follows:

$$h \leq 903{,}844 \rightarrow LegacyDifficulty$$
$$h \geq 903{,}845 \rightarrow ASERT$$

This branch has the following properties:
· it is not changed by runtime configuration
· it is not changed by external signaling or governance
· it is determined by fixed constants defined in consensus parameters

Therefore, for blocks at the same height, all nodes select the same difficulty computation path.

9.2 Subsidy Path and Consensus Enforcement at ConnectBlock

Subsidy for a normal block is computed as:

$$Subsidy(h) = BaseSubsidy(h)$$

At the redistribution block H_r, the following applies:

$$Subsidy(H_r) = BaseSubsidy(H_r) + 630{,}000$$

This subsidy is enforced at the block-connection stage (ConnectBlock).
Consensus condition:

$$CoinbaseOutput \leq Subsidy(h) + \sum Fees$$

where:
· CoinbaseOutput: total coinbase outputs
· $\sum Fees$: sum of all transaction fees in the block

If coinbase outputs exceed the maximum allowed value, the block is invalid under consensus.
This validation is performed at the consensus level, not at the policy level.

9.3 Immutability of Consensus Constants

The following values are consensus constants:
· Halving Interval = 210,000
· Target Block Time = 300 seconds
· ASERT Half-life = 172,800 seconds
· powLimit
· Max Serialized Block Size = 32MB

These values are part of the protocol definition and cannot be changed by runtime options or policy variables.
If changed, consensus incompatibility with the existing chain occurs and a new chain fork is formed.
That is, the above constants are not configuration values but consensus-invariant elements.

10. UTXO State Transition Model

10.1 State Composition

The consensus state at block height $h$is defined as:

$$UTXO_h = UTXO_{h-1} - Spent_h + Created_h$$

where:

· $Spent_h$: set of all outputs spent in block h

· $Created_h$: set of all outputs newly created in block h

UTXO is defined as a pure set, and each element has the following structure:

$$UTXO = \{(txid, index, value, scriptPubKey)\}$$

State-transition properties:

· the state is deterministic

· the state depends only on the previous state and the current block

· the state does not depend on external inputs or runtime environment

This model preserves Bitcoin's UTXO-based state transition.

10.2 Law of Value Conservation

For normal transactions, the following conservation relation holds:

$$\sum Inputs = \sum Outputs + Fees$$

This means:

· arbitrary creation of value is impossible

· arbitrary destruction of value is impossible

· all value movement is conserved within consensus rules

Coinbase transactions are an exception and must satisfy:

$$CoinbaseOutput \leq Subsidy(h) + \sum Fees$$

If the allowed subsidy limit is exceeded, the block is invalid under consensus.
This validation is performed at the consensus code level, not at the policy level.

10.3 Double-Spend Prevention Condition

All transaction inputs must satisfy:

$$Input \in UTXO_{h-1}$$

That is, each input must be an unspent output existing in the prior state.
As soon as an output is spent, the corresponding entry is removed from the UTXO set.

Therefore, the same output cannot be spent twice.

This structure guarantees:
· no double-spend within a block
· no double-spend across blocks
· consistency maintained via state recomputation during chain reorganization (reorg)

10.4 Atomic State Application

Block application is performed as an atomic operation:

$$State' = \begin{cases} Apply(Block_h) & \text{if Valid} \\ Unchanged & \text{if Invalid} \end{cases}$$

Partial application is not permitted.
If validation fails during processing, the state remains as the previous state.

This structure guarantees:
· state consistency maintenance
· complete restoration on failure
· deterministic results even under parallel environments

VI. Chain Selection and Security Model

11. Definition of Accumulated Work

The work of an individual block is defined as:

$$Work(h) = \left\lfloor \frac{2^{256}}{Target(h) + 1} \right\rfloor$$

where:
· $Target(h)$is the consensus difficulty target for the block
· all computations are performed within 256-bit integer range

The $+1$term ensures integer-division boundary stability and prevents exceptions in special boundary cases such as Target = 0.
Because work is proportional to the inverse of Target, as Target decreases, Work increases.

11.1 Accumulated Work

Accumulated work of a chain is defined as:

$$ChainWork(h) = \sum_{i=0}^{h} Work(i)$$

This is not block count, but the integer sum of valid work performed across blocks. Accumulated work computation is also performed within 256-bit integer range.

11.2 Active Chain Selection Rule

The active chain is defined as:

$$ActiveChain = \arg\max ChainWork$$

That is, the chain with the highest accumulated work is adopted as the consensus chain.

This selection has the following properties:
· it does not depend on block count
· it automatically reflects difficulty variation
· it preserves the same criterion before and after the fork

Chain selection is not a simple length comparison, but a comparison of total accumulated computation.

11.3 Meaning for the Security Model

This structure means:
· consensus is protected by computational cost
· only a chain with more computational resources expended can be selected
· simple length extension using low-difficulty segments does not guarantee consensus superiority

Therefore, BTCBT chain selection is directly anchored to the Proof-of-Work security model.

12. Reorg and Attack Model

12.1 Chain Fork and Reorganization

For two chains A and B, the active chain is selected by the following criterion:

$$ChainWork_A > ChainWork_B \Rightarrow A \text{ selected}$$

That is, the chain with greater accumulated work becomes the active chain.

If accumulated work is equal:
· temporarily keep the first received chain
· re-evaluate based on ChainWork when subsequent blocks arrive

Chain selection is determined by accumulated work rather than block count, and this rule is applied identically both before and after the fork.

State handling during a reorg
If chain reorganization occurs, a node performs the following procedure:

1. Roll back the existing active chain to the fork point
2. Restore the UTXO state to the state before that height
3. Apply the blocks of the new chain sequentially

State transition is defined by:

$$State_h = Apply(Block_h, State_{h-1})$$

This process is deterministic, and:
· uses only block data and previous state as inputs
· policy logic does not intervene in consensus judgment
· state is always recomputable

Therefore, consensus integrity is preserved during reorg.

12.2 51% Attack Condition

The attack success condition is:

$$ChainWork_{attacker} > ChainWork_{honest}$$

This means that the attacker must generate more accumulated work than the honest network.

The attacker's success probability is expressed, for attacker hash ratio $q$and confirmation depth z, in the same form as Bitcoin's probability model:

$$P(success) \approx \sum_{k=0}^{\infty} C(z+k-1,k) \cdot q^{z+k} \cdot (1-q)^k$$

where:

· z: number of confirmed blocks

· q: attacker's hash ratio

· $C(n,k)$: combination coefficient

When $q < 0.5$, as $z$increases, the probability of attack success decreases exponentially. This model follows the same PoW security assumptions as Bitcoin.

12.3 Difficulty Adaptation Structure Based on ASERT

The legacy 2016-block adjustment method forms a fixed window.
BTCBT uses the ASERT algorithm:

$$Target(h) = T_0 \cdot 2^{\Delta t/172{,}800}$$

Here, half-life is 172,800 seconds (2 days).

Characteristics of this structure:

· difficulty adjustment is applied continuously on a per-block basis

· there is no adjustment-delay window

· under abrupt hashrate changes, exponential correction applies immediately

Therefore, prolonged difficulty gaps do not structurally occur.

VII. Definition of Consensus Parameter Constants

All consensus constants are fixed at compile time.

· Fork height = 903,844

· Halving interval = 210,000

· Target block time = 300 seconds

· ASERT half-life = 172,800 seconds

· powLimit

· Max serialized block size = 32MB

· Block weight limit = 4,000,000

· Coinbase maturity = 100

These values are part of the consensus code and cannot be modified by runtime settings, policy options, or node configuration changes.

13.1 Definition of powLimit

powLimit defines the maximum permitted Target value:

$$0 < Target(h) \leq powLimit$$

Meaning:
· difficulty cannot be relaxed beyond this value
· the compact representation and ASERT computation result must always satisfy this range
· any Target exceeding powLimit is invalid under consensus

This is an upper-bound condition that structurally limits difficulty from becoming arbitrarily low.

13.2 Coinbase Maturity

Coinbase outputs must satisfy:

$$SpendableHeight \geq h + 100$$

where $h$is the height of the block containing the coinbase.

Meaning:
· mining rewards are spendable only after at least 100 blocks
· ensures stability of reward rollback under reorg
· provides a security buffer under short chain-split conditions

This value is also a consensus condition, not a policy condition.

13.3 Consequences of Changing Constants

If any of the above consensus constants is changed, the following occurs:
· Target computation results change
· block validity criteria change
· ChainWork computation results change

As a result, consensus incompatibility with the existing chain occurs, forming a new chain split.
Therefore, the above constants are not configuration values but part of protocol definition and consensus invariants.

VII. Separation of Unchanged and Changed Domains

14. Unchanged Consensus Elements

The following consensus structures are maintained identically to Bitcoin:
· 80-byte block header structure
· SHA-256d (Double SHA-256) proof-of-work hash algorithm
· UTXO-based state transition model
· Script execution semantics
· SegWit validation path
· Taproot validation structure
· ChainWork accumulated work computation method
· MedianTimePast-based time constraints
· SigOps cost limiting structure

The above items are core consensus elements that constitute block validation, state transition, and chain selection. BTCBT does not modify these mechanisms.

14.1 Restriction of Change Scope

BTCBT changes are limited to the following domains:
· difficulty adjustment algorithm (ASERT adoption)
· certain consensus constants (e.g., target block interval, block size upper bound)

The fundamental consensus framework (state model, validation order, chain selection principle) is maintained.
Therefore, this hard fork is not a redesign of the consensus framework, but a deterministic branch that modifies specific mathematical and constant parameters while preserving the existing structure.

15. Changed Consensus Elements

BTCBT changes the following consensus elements:
· introduction of ASERT difficulty adjustment algorithm
· change of target block interval from 600 seconds to 300 seconds
· increase of maximum serialized block size from 1MB to 32MB
· application of the 630,000 BTCBT subsidy redistribution structure
· halving interval remains 210,000 blocks; due to the shortened block interval, the real-time cadence converges to an approximately 2-year cycle

These changes are limited to the following purposes:

· continuous difficulty adaptation under hashrate variability
· removal of fixed-window adjustment delay
· increased block processing capacity
· redistribution of issuance timing structure

15.1 Structural Limits of Changes

BTCBT does not change the following core consensus structures:
· proof-of-work algorithm (SHA-256d)
· UTXO-based state transition model
· ChainWork-based chain selection rule
· Script execution semantics
· SegWit and Taproot validation structures

Therefore, this hard fork is not a redesign of the consensus framework, but a deterministic modification of the difficulty adjustment method and certain consensus constants.

15.2 Supply Cap Invariance Condition

The following supply invariant condition is maintained:

$$TotalSupply \leq 21{,}000{,}000$$

The 630,000 BTCBT is not new inflation, but time redistribution within the total supply cap.
Because the amount paid in advance at the redistribution block is decremented by the same amount afterward:

$$\Delta TotalSupply = 0$$

The 21,000,000 BTCBT supply cap is not changed.

VIII. Summary of Consensus Invariants

BTCBT consensus must always satisfy the following invariants for all block heights h.
These conditions define consensus validity, supply consistency, and determinism of chain selection.

16.1 Proof-of-Work Validity

$$Hash_{SHA256d}(Header_h) \leq Target(h)$$

· every block must include valid proof-of-work
· $Target(h)$is determined by the Legacy or ASERT formula depending on height
· blocks failing the condition are invalid under consensus

16.2 Supply Cap Preservation

$$TotalSupply(h) \leq 21{,}000{,}000$$

TotalSupply includes cumulative block subsidies.
The 630,000 BTCBT redistribution satisfies:

$$+630{,}000 - 630{,}000 = 0$$

Therefore, the long-term total supply cap does not increase.

16.3 Monotonic Decrease of Subsidy

For halving step k:

$$Subsidy_{k+1} \leq Subsidy_k$$

The halving structure preserves stepwise monotonic decrease.
The redistribution block includes a short-term subsidy variation, but the supply path consistency is preserved.

16.4 Monotonic Increase of Accumulated Work

$$ChainWork(h) > ChainWork(h-1)$$

Individual block work is defined as:

$$Work(h) = \left\lfloor \frac{2^{256}}{Target(h)+1} \right\rfloor$$

Because $Work(h)$is always positive, accumulated work increases whenever a block is added.

16.5 Timestamp Constraints

$$MTP(h-1) < Timestamp(h) \leq NetworkTime + 7200$$

This ensures:

· prevention of backward time movement

· prevention of future timestamps beyond 2 hours

· stability of inputs for difficulty computation

16.6 Target Consistency

$$Expand(nBits_h) = Target(h)$$

The compact representation in the header must match the computed consensus target; otherwise the block is invalid.

16.7 Determinism of Consensus Decisions

Consensus judgment depends only on the following inputs:

$$Consensus = F(Block_h, State_{h-1}, Constants)$$

Constants include:

· Fork height = 903,844

· Halving interval = 210,000

· Target block time = 300 seconds

· ASERT half-life = 172,800 seconds

· powLimit

· block size and weight upper bounds

Consensus judgment does not depend on external signaling, node settings, policy variables, or majority voting.