WHITE PAPER

Unified Networks across Plaintext and Tensorlock Domains

White paper · October 2026

Summary

Tensorlock encrypts training, test, and inference data so that a fully connected artificial neural network (ANN) can train on the ciphertext directly. A network trained under one Tensorlock key works only on data encrypted with that key. This paper describes how one network can serve several domains: plaintext data, and data encrypted under one or more Tensorlock keys.

Two designs are central.

A **routed ensemble** keeps one complete network for each domain and sends each input to the matching network. It is simple, and its networks share nothing.

A **shared-trunk network** gives each domain its own input layer, called a domain adapter, and shares all later layers. Each domain then benefits from the data of the others. The design follows from the mathematics of Tensorlock, which confines the effect of the key to the first layer.

Security depends on what links the domains. Independent datasets under independent keys give an attacker no relation between the keys. Shared examples, a plaintext domain, or closely matched statistics create such a relation. A shared-trunk network can also encode it.

This paper is conceptual. It reports no experiments, and it makes no security claim beyond those in the initial Tensorlock manuscript.

1. Background

1.1 The Tensorlock transformation

Tensorlock applies four steps to each example: padding, perturbation, fixed index shuffling, and random orthogonal matrix transformation. Let $x \in \mathbb{R}^d$ be a plaintext vector. Padding and perturbation produce a vector of higher dimension $D > d$:

$$u = \mathrm{Pert}(\mathrm{Pad}_{S,r}(x)) \in \mathbb{R}^D.$$

Here, S is the hidden set of coordinates that hold the plaintext signal, and r is a padding vector drawn separately for each example. A fixed permutation matrix P and a Haar-random orthogonal matrix Q then give the ciphertext:

$$z = uA, \qquad A = PQ, \qquad A^\top A = I.$$

The key comprises S, A, and the settings that generate padding and perturbation. A key holder can reverse the orthogonal and padding steps:

$$\hat{x} = \left(zA^\top\right)_S = x + \eta.$$

The residual η is the perturbation noise. The key holder recovers x only up to this noise.

The orthogonal step preserves inner products and distances among the padded vectors. The ciphertext therefore discloses the geometry of the padded dataset, and it conceals the coordinate basis.

1.2 Parties and threat model

This paper uses the threat model of the initial Tensorlock manuscript. Party A owns a plaintext dataset and a key. Party B receives the encrypted dataset and labels, and trains the network. Party C is an attacker who holds the encrypted dataset and labels, and who may be part of Party B.

That manuscript also defines an **inference oracle** controlled by Party A. After training, Party B passes the trained network to Party A. The inference oracle

accepts a plaintext example, encrypts it privately, runs the trained network, and returns only the result. It never discloses ciphertext for a submitted plaintext.

This paper extends the setting to several data owners, $A_1, \ldots, A_K$. Each owner holds its own dataset and its own key, and each operates its own inference oracle. Party B receives every encrypted dataset and trains one network for all of them.

1.3 Domains

A **domain** is a set of inputs that share one coordinate basis. Domain k contains the data encrypted under key k:

$$Z_k = U_k A_k, \qquad k = 1, \ldots, K.$$

Plaintext data form domain 0. The plaintext domain behaves as a domain whose key is public.

Networks trained in different domains solve the task in different coordinate bases. Their weights do not correspond, even when their architectures are identical. Averaging the weights of two such networks therefore does not produce a working network.

The domain of every input is known. Each encrypted dataset arrives from an identified owner, and each inference oracle uses its own key. A unified network can therefore route each input by its domain and need not infer it.

1.4 The first-layer relationship

The key affects a fully connected network only through its first layer. Let a plaintext network begin with

$$a = \phi(xW + b),$$

where W is the first weight matrix and ϕ is the activation. Let Π_{S_k} be the $D \times d$ matrix that selects the signal coordinates of key k. The first-layer matrix

$$W_k = A_k^{\top} \, \Pi_{S_k} \, W$$

then gives

$$zW_k = \hat{x}W = (x + \eta)W.$$

A network for domain k with this first layer reproduces the plaintext network, up to the perturbation noise. All later layers are identical. In principle, networks for different domains need differ only in the first layer.

This relationship has two consequences. It motivates the shared-trunk network in Section 3. It also means that two first layers that compute the same function in two domains disclose the transformation between those domains. Section 5 returns to this point.

The relationship shows that a matching first layer exists. Training without the key must find one from data.

2. Routed ensemble

2.1 Method

The routed ensemble keeps one complete trained network f_k for each domain. A switch sends each input to the network of its domain:

$$f(x) = f_k(x) \quad \text{for an input from domain } k.$$

Party B trains each network separately, as the initial Tensorlock manuscript describes. No further training is needed.

A learned gate can replace the switch when the domain of an input is unknown. The gate estimates the domain from the input and weights the outputs of the networks:

$$f(x) = \sum_{k=0}^{K} g_k(x) \, f_k(x).$$

A gate is feasible, because the domains differ in dimension or in the distribution of coordinate values. It identifies the domain and does not recover a key. In the settings considered here, the domain is known, and the switch suffices.

2.2 Properties

Each domain keeps the accuracy of its own network. The ensemble adds no interaction, and it places no new information in any one network.

The networks share no parameters. No domain benefits from the data of another, and the total size grows in proportion to the number of domains.

3. Shared-trunk network

3.1 Architecture

The shared-trunk network gives each domain its own first layer and shares every later layer. The first layer of domain k is its **domain adapter**:

$$T_k : \mathbb{R}^{D_k} \to \mathbb{R}^w, \qquad T_k(x) = \phi(xW_k + b_k).$$

The shared later layers form the **trunk**, C. For an input from domain k,

$$f(x) = C(T_k(x)) \,.$$

The domains can differ in ciphertext dimension D_k, because each adapter maps its own input to the common width w. An adapter can also contain more than one layer.

3.2 Training without paired examples

Two procedures train the network. Neither needs the same example in two domains.

Joint training. Party B trains all adapters and the trunk together on the labeled data of every domain. Each example passes through the adapter of its domain. The shared labels supply the common objective.

Adapter fitting. Party B starts from a trained network for one domain and fixes its trunk. For each further domain, Party B trains only a new adapter on the data of that domain. The first-layer relationship shows that a suitable adapter exists when the domains describe the same kind of data. Adding a domain does not require the data of any other domain.

Adapters taken unchanged from separately trained networks do not fit a common trunk. Separately trained networks order and scale their hidden units differently. One of the two procedures above is therefore required.

3.3 Size and shared learning

Each further domain adds one adapter, not one network. The saving depends on the architecture. In the fully connected network of the initial Tensorlock manuscript, the first layer holds about three quarters of the parameters. The saving is small for that network and larger for deeper networks.

The principal benefit is shared learning. The trunk trains on the pooled data of every domain. A data owner with a small dataset can then gain from the data of the others, and no owner discloses plaintext or a key.

3.4 Inference

Each data owner needs only its own path. Party B passes adapter T_k and the trunk to owner A_k. The inference oracle of that owner encrypts each submitted example, runs the path, and returns only the result.

Party B need not distribute the adapters of other domains. Plaintext inputs use adapter T_0 and need no oracle.

4. Other approaches

4.1 Weight-space merging

Two trained networks can be merged directly in weight space. The hidden units of one network are first reordered to match those of the other. The matched weights are then interpolated:

$$W_3^{(l)} = \alpha W_1^{(l)} + (1-\alpha)\widetilde{W}_2^{(l)}.$$

The merged network is then trained further on data from both domains. This approach needs no architecture change. It faces a difficulty at the first layer, where the two networks operate in unrelated bases and no reordering of hidden units reconciles them. It is the least promising of the approaches described here.

4.2 Paired-example alignment

Alignment methods train on pairs: the same example in two domains. For a pair (x_i, z_i), a loss term draws the two hidden representations together:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda \left\| T_0(x_i) - T_k(z_i) \right\|_2^2 .$$

A related method, distillation, trains the path of one domain to reproduce the hidden representations of a network trained in another. Both methods produce a representation that is common to the domains.

Both methods require a party that holds the same examples in two domains. Section 5.2 shows that such pairs disclose the transformation between the domains. Only a party trusted with both keys should apply these methods.

4.3 Direct conversion by a key holder

A party that holds the keys needs no training to move a network between domains. It computes the new first layer from the relationship in Section 1.4. A data owner that replaces key 1 with key 2, and keeps the padding layout, converts the first layer W_1 as follows:

$$W_2 = A_2^{\top} A_1 W_1.$$

Conversion is exact and immediate. The converted network should remain with the key holder, for the reason given in Section 5.4.

5. Security considerations

A unified network brings several domains before one trainer. The initial Tensorlock manuscript analyzes one encrypted dataset under one key. This section considers what the combination adds.

5.1 Independent datasets under independent keys

Consider the idealized case with orthogonal transformation only. An attacker observes $Z_k = U_k A_k$ for each of K domains. For any orthogonal matrices $R_1, \ldots, R_K$, the alternative datasets and keys

$$U'_k = U_k R_k, \qquad A'_k = R_k^\top A_k$$

produce the same observations, because $U'_k A'_k = U_k A_k$. Each R_k can be chosen independently. The observations therefore leave a separate orthogonal ambiguity for every domain.

The **relative key** between two domains is $A_1^\top A_2$. Under the alternatives above, it becomes $A_1^\top R_1 R_2^\top A_2$, which takes any orthogonal value. The observations alone do not determine the relative key.

Additional encrypted datasets therefore add no algebraic constraint on any one key. Each new domain brings one more unknown basis and one more unknown dataset. Only knowledge that relates the datasets to each other can reduce the ambiguity.

5.2 Bridges between domains

A **bridge** is any such relation. Four kinds of bridge are important to consider.

Shared examples. If the same padded example appears under two keys, then $z^{(2)} = z^{(1)} A_1^\top A_2$. A sufficient number of such pairs determines the relative key by linear regression.

Known plaintext. A pair of plaintext and ciphertext for one example is a shared example between the plaintext domain and an encrypted domain. Enough

such pairs determine the map $A_k^\top \Pi_{S_k}$ from ciphertext to plaintext, up to perturbation noise.

Matched statistics. The initial Tensorlock manuscript finds that recovery becomes possible when an attacker holds exact or near-exact plaintext covariance. Two domains with closely matched covariance offer the same anchor for alignment with each other. That manuscript also finds that recovery degrades rapidly as the match weakens.

Shared labels. A unified network presumes a common task, so the domains share a label set. If the examples of each class are distributed alike in two domains, the class averages correspond across them. These correspondences constrain the relative key in a small number of directions.

The last two bridges are in tension with the purpose of unification. Datasets are worth combining for one task when they resemble each other. The more closely they resemble each other, the stronger the statistical bridge.

5.3 The value of a relative key

A relative key translates ciphertext of one domain into the coordinates of another. Alone, it decrypts neither domain.

It becomes a decryption key in two cases. In the first, the attacker holds the key of one of the two domains, as a data owner allied with the attacker would. In the second, one of the two domains is plaintext, and the relative key is then the key itself.

A plaintext domain therefore deserves particular care. Plaintext data that resemble a private dataset also supply the covariance knowledge that the initial Tensorlock manuscript identifies as the route to recovery. This exposure exists before any unification.

5.4 The unified network as a channel

A shared-trunk network can itself encode a relative key. Suppose two adapters compute the same hidden function, so that $zW_k \approx xW_0$ for corresponding inputs. A party that holds both adapters can then compute

$$\hat{x} = z\, W_k\, W_0^{+},$$

where W_0^{+} is the pseudoinverse of W_0. The result recovers the plaintext, except in the directions that the first layer discards.

Three situations produce such adapters. Paired-example alignment trains for this condition directly. Direct conversion by a key holder satisfies it exactly. Training without pairs does not require it, and the extent to which it arises is unanalyzed.

A common hidden representation offers a weaker route of the same kind. A party that holds two paths can search one domain for an input whose hidden representation matches that of a ciphertext from another. How much example-specific information this search recovers is also unanalyzed.

Three measures limit these channels. The routed ensemble shares no parameters and creates no common representation. Party B can distribute to each owner only its own path, as Section 3.4 describes. A key holder can keep converted networks and paired data inside its own environment.

Party B holds every adapter during training. These measures therefore do not protect against an adversarial Party B.

6. Comparison

Approach	Shared parameters	Paired examples required	Growth per added domain	Security note
Routed ensemble	None	No	One network	No new channel
Shared trunk, joint training	Trunk	No	One adapter	Adapters may encode a relative key; unanalyzed
Shared trunk, adapter fitting	Trunk	No	One adapter	Adapters may encode a relative key; unanalyzed
Weight-space merging	All	No	None	Unanalyzed
Paired-example alignment	Trunk	Yes	One adapter	Pairs disclose the relative key
Direct conversion	All but first layer	No; requires keys	One first layer	Converted layer discloses the relative key

The routed ensemble gives up shared learning and adds no exposure. The shared-trunk network gains shared learning and requires an analysis of what its adapters disclose.

7. Limitations

This paper describes designs and reports no measurements. The accuracy of a shared-trunk network across Tensorlock domains is untested. The first-layer relationship shows that suitable adapters exist, but it does not prove that training finds them.

The argument in Section 5.1 covers the orthogonal step alone and assumes no relation among the datasets. Section 5.2 lists relations that weaken it. The size of each effect has not been measured.

The security considerations identify exposures and do not bound them. Attacks on other data-encoding methods have shown that such transformations can contain structure that initial analysis misses.

The initial Tensorlock manuscript does not claim information-theoretic privacy or general ciphertext indistinguishability, and that limitation applies to the designs discussed here.

8. Conclusion

One network can serve plaintext data and data under several Tensorlock keys. The key reaches a fully connected network only through its first layer. A network with one adapter for each domain and a shared trunk therefore suits the problem, and it lets each domain learn from the data of the others.

The security of the combination depends on the bridges between domains. Independent datasets under independent keys give an attacker no relative key. Shared examples, a plaintext domain, and closely matched statistics each supply one, and a shared-trunk network may encode it. The routed ensemble avoids this exposure at the cost of shared learning.